UNIVERSIDAD
DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

TESIS DOCTORAL

Mejoras en tratamiento de problemas de clasificación con modelos basados en autoencoders

Autor:
Francisco David Charte Luque
<fdavidcl@ugr.es>

Directores:
Francisco Herrera Triguero
Francisco Charte Ojeda

Granada, mayo de 2022

UNIVERSIDAD
DE GRANADA

Mejoras en tratamiento de problemas de clasificación con modelos basados en autoencoders

Tesis para la obtención del título de doctor por la Universidad de Granada
Programa de doctorado en Tecnologías de la Información y la Comunicación

Francisco David Charte Luque
<fdavidcl@ugr.es>

Directores de tesis

Dr. D. Francisco Herrera

Soft Computing y Sistemas de Información Inteligentes (Universidad de Granada)

Dr. D. Francisco Charte

Sistemas Inteligentes y Minería de Datos (Universidad de Jaén)

Editor: Universidad de Granada. Tesis Doctorales
Autor: Francisco David Charte Luque
ISBN: 978-84-1117-423-7
URI: http://hdl.handle.net/10481/75933

Mejoras en tratamiento de problemas de clasificación con modelos basados en autoencoders
Francisco David Charte Luque

Funding
This doctoral thesis was funded by the predoctoral program Formación del Profesorado Universitario (ref. FPU17/04069) from the Spanish Ministry of Universities.

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

It's the questions we can't answer that teach us the most. They teach us how to think. If you give someone an answer, all they gain is a little fact. But give them a question and they'll look for their own answers.

– Patrick Rothfuss, The Wise Man's Fear

Machines take me by surprise with great frequency.

– Alan Turing

Acknowledgments

I hope the reader appreciates the hard work that has ultimately produced this thesis, which is the result of the direct and indirect effort of many people.

First and foremost, I am infinitely grateful for the opportunities, guidance, help and trust my supervisors have offered me. I have been extremely lucky to have such passionate, knowledgeable and caring people directing this work. These four years have allowed me to be creative, to learn much more and to go beyond even what I expected. Thank you, dad. Thank you, Paco.

Learning is one of my passions. I am thankful for all the good teachers I have had, they are the motivation that students need to strive for the best. I want to take a moment to thank and remember two of them who sadly are no longer with us. With Manuel Arroquia I learned not to accept something as truth without finding a reasoning. Julio Ortega showed me that there is joy in teaching and caring about your students. The curiosity you inspired in your students and the aspiration for excellence you inspired in your colleagues live on.

Big thanks to my friends in the SCI2S group and at the CITIC for making these years fun, making me see things from other perspectives and being there to support each other.

It is fairly uncommon that one of your supervisors also happens to be your father. Because of this, the rest of my family had to bear with many boring conversations at home when work topics inevitably came up. Still, they have always been there to share a good time with us. Mom, Alex, thank you for your endless patience and support.

Finally, I have to express my gratitude to the person who has always been next to me during these four years, who has lifted my spirits more times than I could count and has shown me nothing but love. Thank you from the bottom of my heart, Esperanza.

This work is a reality thanks to all of you.

Abstract

Improvements in treatment of classification problems with autoencoder-based models

The present doctoral thesis tackles the study and application of a specific tool from the data science field, autoencoders, which are artificial neural networks able to transform the variable space of a dataset according to a certain selected criterion. Manipulating and transforming variables is a crucial task in data mining, since it can largely determine the complexity of a data analysis problem and, as a result, affect the behavior of learning methods which are used to extract useful knowledge. Moreover, the recent surge in data collection and processing for all kinds of purposes causes these tasks to be less and less feasible to be performed by hand, so there is a need of automatic methods to solve them.

Autoencoders are models that belong to the field of representation learning, and can be much more flexible and adaptable than other, more classical methods such as principal component analysis. This versatility has been studied via a thorough analysis of their inner workings and the different varieties of models than can be created based on their essential components. As a complement, a new software tool has been developed to provide easy access to these models and eliminate an important existing knowledge barrier which could prevent their use.

An extensive search has been conducted in the literature for problem typologies whose difficulty is related to the data representation, so as to open the possibility for autoencoder-based solutions. Datasets can present several issues: those linked to the very structure of the data points, like the use of several objects to describe a sole instance; those relative to the complexity of categorized data, or tasks that do not provide additional information and must be solved by means of feature analysis.

With the aim of creating a novel contribution in the field of autoencoders, three new models have been developed to tackle the problem of complexity in categorized data. They are able to simplify the borders between categories in order for a classification method to improve its performance.

In summary, the main contributions of this thesis are as follows:

- A **theoretical analysis and taxonomy** of the main autoencoder variants present in the literature, composing a guide to ease their selection and use.
- A complete **software package** which automatizes a great part of the implementation work for autoencoders and simplifies its use to a level similar to other feature extraction methods.
- A **synthesis and organization** work of the peculiarities that supervised learning problems can present when data points are represented in a nonstandard fashion.
- A **demonstration of the diverse applications** of autoencoder-based models, identifying and exposing several strategies to solve unsupervised problems by means of variable transformations.
- Three new models, **Scorer, Skaler and Slicer**, focused on data complexity reduction in classification problems.

This document introduces all global concepts needed to understand the published articles and provides a theoretical vision of the representation learning problem and of the deep learning tool set, which includes the main object of study. In addition, it explains the techniques that help put into practice these models and how they execute on computation infrastructures. Next, the material published throughout the doctoral period is introduced and five articles published in renowned journals are reproduced. Finally, these and other activities carried out are summarized and the lines of work that would continue the achieved advancements are presented.

Resumen

Mejoras en tratamiento de problemas de clasificación con modelos basados en autoencoders

La presente tesis doctoral aborda el estudio y aplicación de una herramienta particular del ámbito de la ciencia de datos, los *autoencoders*, que son redes neuronales artificiales capaces de transformar el espacio de variables de un conjunto de datos según un criterio escogido. La manipulación y transformación de variables es una tarea crucial en minería de datos, puesto que puede determinar en gran medida lo complejo que resulte un problema de análisis de datos y, por tanto, afectar al comportamiento de los métodos de aprendizaje con los que se pretende extraer conocimiento útil. Además, el reciente incremento en recolección y procesamiento de datos para todo tipo de propósitos propicia que cada vez menos tareas de transformación se puedan realizar manualmente, por lo que son necesarios métodos automáticos que las resuelvan.

Los *autoencoders* son modelos que se encuadran en el campo del aprendizaje de representaciones, y resultan mucho más flexibles y adaptables que otros métodos más clásicos como el análisis de componentes principales. Para estudiar esta versatilidad, se ha realizado un análisis pormenorizado de su funcionamiento y de las diferentes variedades de modelos que se pueden crear fundamentándose en sus componentes básicos. Como complemento, se ha construido una herramienta software que proporciona fácil acceso a estos modelos y elimina una importante barrera de conocimiento existente a la hora de utilizar los *autoencoders*.

Asimismo, se ha llevado a cabo una extensa búsqueda en la literatura de tipologías de problemas cuya dificultad esté relacionada con la representación de los datos, de forma que se pueda plantear una solución basada en *autoencoders*. Se han identificado varias clases de conflictos que pueden presentar los conjuntos de datos: los que residen en la propia estructura de los datos como, por ejemplo, el uso de varios objetos para representar una sola instancia; los relacionados con la complejidad de los propios datos cuando están categorizados, o tareas que no aportan información adicional y han de resolverse por medio del análisis de las características.

Con el objetivo de aportar una contribución novedosa al campo de los *autoencoders*, se han desarrollado tres modelos que abordan el problema de la complejidad de los datos categorizados, siendo capaces de simplificar las fronteras entre las categorías de forma que un método de clasificación mejore su rendimiento.

En resumen, las principales contribuciones de la tesis son las siguientes:

- Un **análisis teórico y taxonomía** de las principales variantes de *autoencoders* presentes en la literatura, componiendo una guía para facilitar la selección y el uso de las mismas.
- Un completo **paquete software** que automatiza gran parte del trabajo de implementación de *autoencoders* y acerca su uso a un nivel comparable al de otros métodos de extracción de características más simples.
- Un trabajo de **organización y síntesis** de las particularidades que pueden presentar los problemas de aprendizaje supervisado cuando los datos están representados de formas no estándares.

- Una **demostración de las diversas aplicaciones** de los modelos basados en *autoencoders*, identificando y exponiendo distintas estrategias para resolver problemas no supervisados mediante manipulación de las variables.
- Tres nuevos modelos, **Scorer, Skaler y Slicer**, enfocados a la reducción de la complejidad de datos en problemas de clasificación.

El presente documento introduce todos los conceptos globales necesarios para entender los artículos publicados y aporta una visión teórica de la problemática del aprendizaje de representaciones y del conjunto de herramientas de aprendizaje profundo, dentro del cual se enmarca el objeto principal de estudio. Además, se explican las técnicas que ayudan a llevar a la práctica estos modelos y cómo se ejecutan sobre las infraestructuras de computación. Posteriormente se introduce el material publicado a lo largo del periodo doctoral y se reproducen cinco artículos publicados en revistas científicas de notable reputación. Finalmente se resumen estas y otras actividades llevadas a cabo, y se presentan las líneas de trabajo que continuarían con los avances ya realizados.

Contents

List of Figures

List of Tables

Fundaments

Introduction 1

Current trends in collection of data are increasing, with more and more human activities producing machine-readable information, such as product reviews [1], posts on social media [2], medical images [3], industrial machinery sensor data [4], and more. Automatic processing of data makes it easier to obtain results fast and saves hours of human labor which can be freed for other purposes or dedicated to tasks which cannot be automated. The speed provided by current computation resources also opens new possibilities for leveraging the available data, achieving extraction and manipulation of information at levels infeasible by human hand.

[1]: Fang et al. (2015), "Sentiment analysis using product review data"

[2]: Balaji et al. (2021), "Machine learning algorithms for social media analysis: A survey"

[3]: Willemink et al. (2020), "Preparing medical imaging data for machine learning"

[4]: Gao et al. (2015), "A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches"

The study of problems, tools and solutions related to data integration, processing and analysis has been recently known as data science [5], a field which overlaps branches of mathematics, statistics and computer science, among other disciplines. Current data science applications are present everywhere, from the most industrial settings to direct final user access, and range from machinery fault detection [6], to medical diagnosis assistance [7], customer loyalty in retail [8] and photograph enhancing [9].

[5]: Dhar (2013), "Data science and prediction"

[6]: Carvalho et al. (2019), "A systematic literature review of machine learning methods applied to predictive maintenance"

[7]: Erickson et al. (2017), "Machine learning for medical imaging"

[8]: Ma et al. (2020), "Machine learning and AI in marketing–Connecting computing power to human insights"

[9]: Wang et al. (2020), "Deep learning for image super-resolution: A survey"

The general objective in a data science problem is to model a real world scenario based on the collected data and use the resulting model to provide some information to the end user, for example, a category or label, a ranking, an association or a transformed version of the original data. This is a process that encompasses all steps from data acquisition, to its preparation, processing and analysis. In this context, a wide variety of computer algorithms are used to manipulate and process data. The study and development of these techniques that allow machines to distill knowledge from data are known as *machine learning*.

1.1 Problem setting

There are several stages that compose the process of solving a data science problem, represented visually in Figure 1.1. A great part of the time spent, usually the longest, consists in preparing and preprocessing the available data in order for the posterior learning techniques to extract the maximum possible amount of information and ensure the new knowledge is valid [10]. There exist several traits

[10]: Domingos (2012), "A few useful things to know about machine learning"

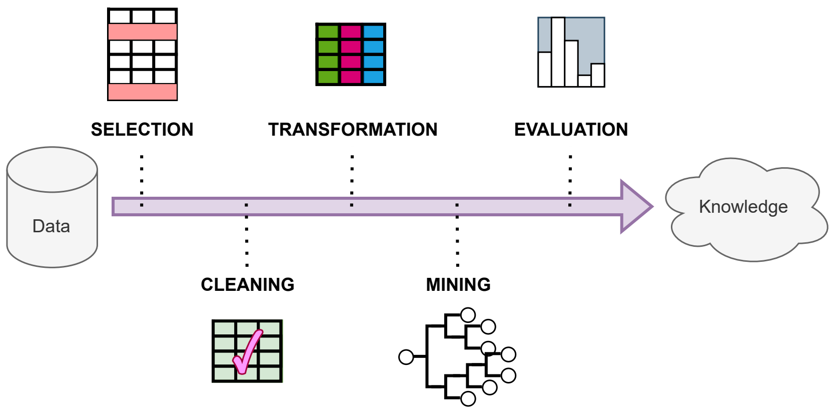


Figure 1.1: Steps of the knowledge discovery in databases (KDD) process ordered from left to right: data selection, cleaning, transformation, mining and evaluation.

of the data that can be manipulated to facilitate the work of learning algorithms: missing values, noisy instances, class imbalance, among others [11].

[11]: García et al. (2015), *Data preprocessing in data mining*

The set of features, that is, the specific representation of each instance in a dataset, is one aspect which machine learning models can be specially sensitive to.

Features that may characterize an event or object adequately for humans may not be ideal for machines to process. For example, a string of text may have meaning for a reader but for the machine it is just a sequence of characters whose semantics are not easily processed in that format. Similarly, an image can be expressed as a series of color values which, if shown on a screen, will display something intelligible for a person's eye, but those values do not hold direct relation to what is contained in the image itself.

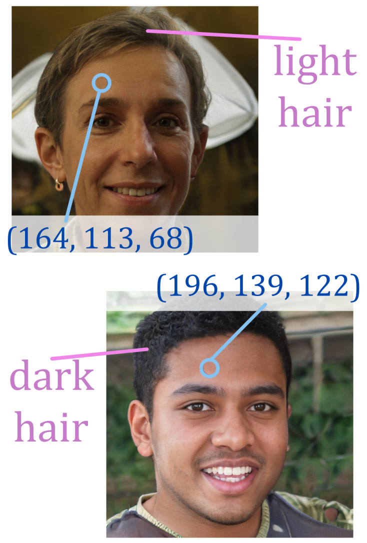


Figure 1.2: Two randomly generated images from `https://thispersondoesnotexist.com`. In blue, available features such as the color values of some pixels. In pink, latent variable values (such as *hair tone*) that are not directly represented in the data but influence those color values.

Furthermore, the techniques used for collecting data can only produce the observable variables in a dataset, but there may be interesting, hidden variables which influence the data in a clearer way. This is a concept known in statistics as *latent variables* [12]. For example, the value "blond" for the latent variable "hair color" might determine the color value of many pixels in an image of a person's face, but those are also influenced by overall lighting and contrast, so the hair color of a person cannot be deduced by just extracting a couple of pixels (see Figure 1.2 for an example). It would be desirable to obtain the meaningful features, but a learning mechanism that takes all observable features into account is needed for these to be extracted.

[12]: Borsboom (2008), "Latent variable theory"

Providing a learning algorithm with unprocessed features usually leads to sub-par performance due to several potential issues: different scales, presence of noisy variables, redundant information, useful information that is obscured and not directly represented, as well as irrelevant information. All of this may confuse a learning method which initially might consider all variables equally relevant.

Other possible obstacles that one may come across when inspecting the features of a dataset are difficult classes, also known as data complexity [13]. This scenario arises when the variables do not provide sufficient information to allow class separation, the relations between variables and the target are highly nonlinear, or other circumstances prevent a learning method from inferring an adequate mapping from the input features to the target variable.

[13]: Cano (2013), "Analysis of data complexity measures for classification"

As a consequence, much time can be spent manually engineering features according to what the practitioner believes the learning method will adapt best. However, this is a process that requires experience and usually involves many attempts at improving results. Instead of this, a new task can be performed by machine learning techniques before the actual knowledge extraction, called *feature learning* or more broadly *representation learning* [14].

[14]: Bengio et al. (2013), "Representation learning: A review and new perspectives"

Feature learning alleviates a notable amount of manual labor and dataset manipulation although, of course, implies that the user know about feature learning methods and their operation. There exists a very diverse array of methods, ranging from simple principal component analysis (PCA) [15] to more complex manifold learning methods [16].

[15]: Jolliffe (1986), *Principal component analysis*

[16]: Lin et al. (2008), "Riemannian manifold learning"

However, not every feature learning method can solve every task. Most of them have a specific criterion that they optimize, such as maximum variance in the case of PCA, point reconstruction from its neighbors as in locally linear embedding [17], or shortest-path distances between points as in Isomap [18]. The objective function of each method determines which behavior will be found in the extracted representation, with little to no room to customize it. There may be situations where it is necessary for the learned variables to adjust to certain properties and, ideally, the feature extraction method should allow to enforce those properties.

[17]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

[18]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

1.2 Tools

The main set of tools that are used to tackle problems along this work is *deep learning*, a subset of machine learning focused on algorithms which extract successive transformations of the data until a solution to the task is reached.

Deep learning originates from the concept of artificial neuron, which takes some inspiration on the biological nervous system [19]. The perceptron, a probabilistic model of the brain, was developed drawing from this abstract neuron [20] and current artificial neural networks are simply generalizations of the multi-layer perceptron (MLP). For a long time, neural networks were very inefficient to train compared to other machine learning methods, and they were not used for many applications as a result.

[19]: McCulloch et al. (1943), "A logical calculus of the ideas immanent in nervous activity"

[20]: Rosenblatt (1958), "The perceptron: a probabilistic model for information storage and organization in the brain."

The resurgence of neural network models for machine learning has happened more recently due to the fact that their optimization had much higher computational cost than other, classical machine learning models. Thus, more efficient optimization strategies in combination with much more powerful hardware (leveraging the computing capacities of modern GPUs) have allowed to train complex deep models which were unfeasible before.

Nowadays, deep learning is used to extract the contents of images, natural language and speech (as well as produce them) in a wide range of applications, from small fitness devices to autonomous vehicles. It is also very present in research fields ranging from medicine to computer security. It powers translation systems [21], search engines [22] and virtual assistants [23].

[21]: DeepL Team (2021), *How does DeepL work?*

[22]: Shu et al. (1999), "A neural network-based intelligent metasearch engine"

[23]: Mycroft.AI (2021), *Mycroft Technologies Overview*

The main advantage of deep learning against traditional machine learning methods is being able to automatically transform raw data into useful representations that depict more complex, high-level concepts. This stage is usually performed manually when approaching machine learning problems with other models.

Since deep learning models perform their own representation learning while training to solve other tasks, one possible use for these models is the extraction of said representations. One can either extract these *deep features* from neural networks which are trained for a different purpose or build a deep learning model dedicated to just learning a new, more useful representation. The latter approach will be our focus during most of this thesis, using a specific category of models known as *autoencoders*.

1.3 Motivation

The questions that we are trying to tackle throughout this thesis can be summarized as follows:

- How can representation learning be approached with deep neural models?
- What benefits can be obtained by transforming data into an appropriate representation?
- Can specific behavior be induced within the transformations, such as separating different classes?

As the trends in usage of deep learning models to solve machine learning problems continue to increase, we focus our interest in their potential to not only tackle supervised problems such as classification, regression or detection, but also wider problems where solutions are not so easily validated, such as feature learning. Since deep learning allows the integration of the feature extraction stage directly within the predictor itself, these types of models should be valuable feature learners for other tasks as well.

Deep neural models dedicated to generating new feature sets could be adapted, as a result, to different purposes. For instance, one could search for feature spaces where "ordinary" data points are very cohesive, and thus anomalous inputs would be easy to identify [24]. Similarly, a transformation of features could allow for better separation of different classes, better distinction between noise and signal, or more meaningful traits that relate to all the original variables.

[24]: Sakurada et al. (2014), "Anomaly detection using autoencoders with nonlinear dimensionality reduction"

Another potential use of feature learning models that catches our interest is the possibility of capturing more than one aspect of each problem instance, which translates to different *views* of the problem, for example, image and text. These views could be processed by different learners or special algorithms, but one could build feature learners that combine the available information into a more machine-ready feature vector.

[25]: Fong et al. (2017), "Interpretable explanations of black boxes by meaningful perturbation"

A current conflict of deep learning models with several areas of interest in machine learning, such as medical applications, is the fact that most are essentially black boxes [25]. This means that the behavior of a trained model is obscured by the intrinsic structure and is thus unintuitive for humans to comprehend. As a result, there is much interest in explaining and justifying the behavior of these models. Enabling the use of interpretable classifiers and regressors such as decision trees by means of better dataset representations can be one way of avoiding black boxes in these contexts.

1.4 Objectives

All the research developed along this work has been revolving around the general aim of improving how autoencoders can achieve a better representation of data in order to extract more reliable models from it.

As a result, the specific objectives that we posed throughout the course of the thesis were the following:

- To acquire a deep understanding on how autoencoders work, how they differ from other feature extraction techniques, and the different approaches to learn new features with them.
- To facilitate the use of autoencoders by unspecialized users, improving on previous implementations in efficiency and ease of use.
- To explore different types of supervised learning problems further from the classical binary (positive/negative) prediction schemes or regression tasks.
- In the context of supervised problems, to develop new ways of optimizing autoencoders so that generated features are able to separate different classes better.

Additionally, there are several tangential issues that were studied due to the interesting relation to the main topic and the opportunity to work on real world problems. These are as follows:

- To identify areas of application where autoencoders are able to provide solutions without the need of additional modelling.
- To evaluate and compare different encoder-decoder architectures for different purposes.
- To analyze whether ensembling several encoder-decoder methods can provide better solutions.

The main objectives outlined above are further detailed in the following subsections.

Didactic resources about autoencoders

Autoencoders are conceptually very different from traditional feature extractors. Unlike these, autoencoders are based on a neural network framework and this allows for a high level of customization and adjustments for each task. However, this availability of diverse options when building an autoencoder makes it less accessible to inexperienced practitioners. This is a barrier that was identified at the start of our research work and, as a result, became an issue we wanted to address.

Our first goal, taking advantage of the usual literature review, was to produce a guide on autoencoders for machine learning users assuming no prior knowledge about neural networks or these models in particular. This guide should cover all the basics in order to be able to grasp what autoencoders compute, how they are trained, how they compare to other feature learners and what options a user may be presented with when choosing to apply this model to their data.

Easy-to-use autoencoder implementations

One of the first obstacles that programmers may come across when working with feature learning tools is that autoencoders are much harder to set up and train than other alternatives like PCA or even complex manifold learning algorithms such as LLE or Isomap, which come already implemented in libraries and can be applied with a simple function call.

In consequence, for a better understanding of autoencoders by the data science community, it is crucial to have access to a software tool which provides all the necessary tooling in order to design, train and use autoencoders without the need to be an expert in this specific area. Our objective was to develop this tool which would provide a simple interface for scientists from other knowledge fields and other practitioners, as well as more complete functionalities for those who need and know how to take advantage of them.

Exploration of supervised problems

With the potential application of autoencoders in mind, but not limiting ourselves to the current state of available strategies and architectures, we aimed to study in detail the diverse range of supervised problems that one may encounter beyond traditional classification (binary or multiclass) and regression (with one output).

The main purpose of this was to identify common properties in different problems that would allow to tackle them from similar perspectives or generalizations of simpler cases.

Development and application of new autoencoder models

After a more general focus on the state of the art, it was important to create new solutions based on what had been learned. These solutions should be applicable as widely as possible and tackle real problems from a novel perspective, leveraging the flexibility of autoencoders.

In order to contribute to the specific field of autoencoders, our aim was to improve on one of the main uses that these models can have, the extraction of better features for classification methods. Instead on relying on the intended classifiers themselves to optimize the quality of the features, like a deep neural network would operate, we opted to choose metrics that inform about the ability of dataset variables to separate different classes. This way, the model is not necessarily focused on the variables that are more relevant to allow a neural network classify, but those that help separate classes in general, which can in turn be useful for various classifiers.

The previous model proposal would be more valuable if useful applications of it were deployed. For a first use, we chose a dataset that was imposing notable levels of difficulty for classifiers to model adequately, the COVIDGR dataset of chest X-ray imaging for COVID-19 detection [26]. This is, as well, an area where a solution involving a simple, interpretable classifier would appeal more to the experts than a black-box model like a neural network.

[26]: Tabik et al. (2020), "COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images"

1.5 Thesis structure

This work comprises several original introductory chapters presenting the necessary theoretical concepts and the basics on how these are put to practice, as well as reproductions of five articles published in peer-reviewed journals, each of them related one or more of the previously established goals for this thesis.

The rest of this document is organized as follows:

- **Chapter 2** describes the basic theory that lies under the subsequent works and the specific models that are developed and used.
- **Chapter 3** explains the practicalities of implementing and training deep learning models, especially autoencoders.
- **Chapter 4** outlines the main results obtained during the research period, providing context and connections among the articles that are reproduced next.

- **Article I**, the first of the journal articles, is a guide on the inner workings of autoencoders, their variants, how to design and how to implement them.
- **Article II** describes into detail the software developed with the objective to facilitate the use of autoencoders.
- **Article III** is a review of the current state of the art in supervised learning further from the standard problems.
- **Article IV** covers the most popular applications of autoencoders apart from classification, with concrete examples and guidelines.
- **Article V** improves the applicability of autoencoders in classification problems by developing new loss functions which help reduce data complexity.
- Lastly, **Chapter 5** draws some conclusions and closing statements on the developed work.

References

[1] Xing Fang and Justin Zhan. “Sentiment analysis using product review data”. In: *Journal of Big Data* 2.1 (2015), pp. 1–14. DOI: 10.1186/s40537-015-0015-2.

[2] TK Balaji, Chandra Sekhara Rao Annavarapu, and Annushree Bablani. “Machine learning algorithms for social media analysis: A survey”. In: *Computer Science Review* 40.100395 (2021), pp. 1–32. DOI: 10.1016/j.cosrev.2021.100395.

[3] Martin J Willemink et al. “Preparing medical imaging data for machine learning”. In: *Radiology* 295.1 (2020), pp. 4–15. DOI: 10.1148/radiol.2020192224.

[4] Zhiwei Gao, Carlo Cecati, and Steven X Ding. “A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches”. In: *IEEE Transactions on Industrial Electronics* 62.6 (2015), pp. 3757–3767. DOI: 10.1109/TIE.2015.2417501.

[5] Vasant Dhar. “Data science and prediction”. In: *Communications of the ACM* 56.12 (2013), pp. 64–73. DOI: 10.1145/2500499.

[6] Thyago P Carvalho et al. “A systematic literature review of machine learning methods applied to predictive maintenance”. In: *Computers & Industrial Engineering* 137.106024 (2019), pp. 1–10. DOI: 10.1016/j.cie.2019.106024.

[7] Bradley J Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L Kline. “Machine learning for medical imaging”. In: *Radiographics* 37.2 (2017), pp. 505–515. DOI: 10.1148/rg.2017160130.

[8] Liye Ma and Baohong Sun. “Machine learning and AI in marketing–Connecting computing power to human insights”. In: *International Journal of Research in Marketing* 37.3 (2020), pp. 481–504. DOI: 10.1016/j.ijresmar.2020.04.005.

[9] Zhihao Wang, Jian Chen, and Steven CH Hoi. “Deep learning for image super-resolution: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2020), pp. 3365–3387. DOI: 10.1109/TPAMI.2020.2982166.

[10] Pedro Domingos. “A few useful things to know about machine learning”. In: *Communications of the ACM* 55.10 (2012), pp. 78–87. DOI: 10.1145/2347736.2347755.

[11] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Vol. 72. Springer, 2015.

[12] Denny Borsboom. “Latent variable theory”. In: *Measurement: Interdisciplinary Research and Perspectives* (2008). DOI: 10.1080/15366360802035497.

[13] José-Ramón Cano. “Analysis of data complexity measures for classification”. In: *Expert systems with applications* 40.12 (2013), pp. 4820–4831. DOI: 10.1016/j.eswa.2013.02.025.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50.

[15] Ian T Jolliffe. *Principal component analysis*. Springer, 1986.

[16] Tong Lin and Hongbin Zha. “Riemannian manifold learning”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.5 (2008), pp. 796–809. DOI: 10.1109/TPAMI.2007.70735.

[17] Sam T Roweis and Lawrence K Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[18] Joshua B Tenenbaum, Vin de Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.

[19] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1016/S0092-8240(05)80006-0.

[20] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), p. 386. DOI: 10.1037/h0042519.

[21] DeepL Team. *How does DeepL work?* https://www.deepl.com/en/blog/how-does-deepl-work. 2021.

[22] Bo Shu and Subhash Kak. “A neural network-based intelligent metasearch engine”. In: *Information Sciences* 120.1 (1999), pp. 1–11. DOI: 10.1016/S0020-0255(99)00062-6.

[23] Mycroft.AI. *Mycroft Technologies Overview*. https://mycroft-ai.gitbook.io/docs/mycroft-technologies/overview. 2021.

[24] Mayu Sakurada and Takehisa Yairi. “Anomaly detection using autoencoders with nonlinear dimensionality reduction”. In: *Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA 2014)*. 2014, pp. 4–11. DOI: 10.1145/2689746.2689747.

[25] Ruth C Fong and Andrea Vedaldi. “Interpretable explanations of black boxes by meaningful perturbation”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 3429–3437. DOI: 10.1109/ICCV.2017.371.

[26] S. Tabik et al. “COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images”. In: *IEEE Journal of Biomedical and Health Informatics* 24.12 (Dec. 2020), pp. 3595–3605. DOI: 10.1109/jbhi.2020.3037127.

Theoretical foundation 2

Machine learning covers a set of problems and tools that overlaps several disciplines. As such, in order to tackle machine learning problems, it is necessary to lay some foundations which help grasp all the concepts involved. Even though this is a rapidly evolving field, there are fundamental topics that remain applicable.

The objective of this chapter is to provide the reader with the definitions, descriptions and examples that allow to understand the rest of this work. We will try to assume little-to-no prior knowledge about machine learning, thus making it as accessible as possible. The following sections go over the basic subjects of machine learning, build the core concepts of deep learning and arrive to the main tools that will be used along the rest of this book, autoencoders.

2.1 Machine learning fundamentals

Machine learning differs from other kinds of computer science disciplines in that its objective is not to give precise instructions for the machine to follow, but instead to provide some form of experience that the machine must learn from in order to extract some information or display some behavior [1]. The algorithms developed for machine learning are essentially mechanisms that take in a certain amount of data, process it and compute the necessary steps to fulfill a specific objetive related to the data. Their output is usually a model, that is, a representation of an approximate solution to the problem.

[1]: Deisenroth et al. (2020), *Mathematics for machine learning*

Data and models

Datum (plural *data*) usually refers to the minimal unit of machine-readable information, for example, the height of a person (numerical value), whether they are an adult or not (binary categorical value), their country of origin (categorical value) or their given name (character string).

Table 2.1: An example dataset describing features of different kinds of animals. Each feature can be numerical (length, legs) or categorical (wings, species).

Length	Legs	Wings	Species
0.40	4	No	Dog
0.01	6	Yes	Fly
1.45	0	No	Dolphin

A *dataset* is a collection of data, usually organized into a table. It contains several *samples*, which correspond to each one of the cases of the problem from which the machine will be able to learn before being presented with new cases. *Variables* are each one of the aspects that have been measured or that characterize each sample. They determine the type of data (character strings, numbers, dates) and the range where values are taken. A usual distinction of numerical variables is between continuous and discrete ones, the former corresponding to real intervals and the latter with a countable set of possible values.

Samples are typically distributed in rows and variables in columns. Table 2.1 shows an example of dataset with 3 samples and 4 variables, where each variable corresponds to a specific kind of data: *length* is a continuous numerical variable, *legs* is discrete numerical, *wings* is binary categorical and *species* is categorical.

A *model* is an abstraction of a dataset that enables the machine to perform the desired operations, for example, generating new data similar to the available, or assigning a category to new data points. A good model should be faithful to the available data, incorporating enough information to describe its behavior and potential relations between variables, so that it can be used as a description of the data and as a tool for solving tasks related to it. Models typically follow some template which includes a range of parameters that can be adjusted in order for the resulting model to represent the data. We will call these templates *untrained models*, whereas the final results will be *trained models*.

Learning and types of learning

In the context of machines learning from data, several types of learning are usually distinguished, according to the feedback that the machine receives while processing data. This concept is known as *supervision*, and usually relates to whether there are available solved cases for the specific problem at hand. A solved case is composed of an input instance and an associated solution or *label*, which may be a numerical value, a categorical value or a more complex structure. Attending to the availability of labels for the learning algorithm, the following learning paradigms are considered [2]:

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

Supervised learning

In a supervised learning setting [3], every observed case of the problem in the dataset is coupled with its solution, so that the machine can learn a mapping out of those associations, from the space of the instances (input space) to the one of the labels (output space). Models generated by learning algorithms in this contexts are usually known as *predictors*, since for each new data point they must guess a label in the output space. For example, in Table 2.1, an appropriate objective task for a supervised learning algorithm is to predict the species of an animal, knowing the rest of its characteristics.

[3]: Caruana et al. (2006), "An empirical comparison of supervised learning algorithms"

Common supervised learning problems are *classification* and *regression*. They differ in the type of output that the predictor must produce: classification implies that the output space is finite, thus the label is just one of a certain number of available *classes*, whereas regression involves guessing a real value from a continuous interval.

Unsupervised learning

The scenario of unsupervised learning [4] covers problems where the solution is not known for the data that is available and, as a result, the model cannot be provided with supervision. Instead, the user of an unsupervised learning algorithm looks to find some sort of inner structure or hidden patterns in the data.

[4]: Celebi et al. (2016), *Unsupervised learning algorithms*

One case where unsupervised learning methods are convenient is when trying to find the most useful variables in a dataset, or even transform the original features onto a more compact set of variables which prevent redundancy and maximize efficiency in relation to information provided per feature. Another typical task is finding associations between items present in the data points, like related articles in a shopping bag or recommended moviess.

Combinations of supervised and unsupervised learning

There are special cases where the task that is being approached is not entirely supervised but not completely unsupervised either, but a mix of both.

The most common combination emerges when the presence of labels in the observed data is mixed, that is, there are instances with associated labels and others with missing labels. This learning paradigm, known as *semi-supervised learning* [5], is of interest in

[5]: Van Engelen et al. (2020), "A survey on semi-supervised learning"

many real-world problems, since annotating each one of the collected data instances with its corresponding label can be costly and time-consuming [6].

[6]: Engelen et al. (2019), "A survey on semi-supervised learning"

Another relevant area that combines predictive and descriptive learning is that of *subgroup discovery* [7], a task where the objective is to find unusual relations (rules) between the input variables and the target variable. Instead of learning to predict this variable, the idea is to identify interesting subsets of instances according to their relation to the target variable and the rest of samples.

[7]: Herrera et al. (2010), "An overview on subgroup discovery: foundations and applications"

Reinforcement learning

A different strategy for machines to learn consists in providing them with positive or negative reinforcements according to their behavior [8]. Instead of providing the algorithm with the complete solution for each problem instance, usually a score is given, evaluating the current solution against some criteria. This learning paradigm fits well with problems where multiple solutions can be acceptable, so the actual solving process is not as important as obtaining the desired result, as well as situations where the aim is to find the most efficient solution. Notable examples of this kind of learning are tabletop games like AlphaGo for Go [9] and even strategy videogames like AlphaStar for StarCraft II [10].

[8]: Kaelbling et al. (1996), "Reinforcement learning: A survey"

[9]: Silver et al. (2016), "Mastering the game of Go with deep neural networks and tree search"

[10]: Vinyals et al. (2019), "Grandmaster level in StarCraft II using multi-agent reinforcement learning"

2.2 Obstacles when learning models

Machine learning models can come across several kinds of difficulties that are relevant to analyze since they are related to the tools and solutions studied in this thesis. Primarily, we will focus on improving the feature sets and tackling certain aspects of supervised learning problems.

Feature sets

The feature set, as explained in Chapter 1, corresponds to the space where each sample takes values. The same events may be expressed by different feature sets according to the information collection procedures. For example, a spoken command may be represented by a precise sound file that was recorded or by the words that were uttered. In the former case, the feature set could be the presence of each frequency at each time point. In the latter, a possible set of features would indicate whether each word from a predefined list was present or not in the sentence.

Traditional algorithms for adjusting models tend to process data "as is", which means that they perform few transformations (or none) to each vector before using them directly to fit model parameters. This causes them to underperform when the representation of the vectors (i.e. the set of features) is not ideal. As a consequence, it is usually convenient to preprocess data beforehand, using one or several tools that will manipulate the features looking to improve the performance of the learning algorithm. This is known as *feature extraction*, feature learning or representation learning [11].

[11]: Bengio et al. (2013), “Representation learning: A review and new perspectives”

Potential issues with supervised problems

Although supervised learning tasks provide the desired answer for all observed cases, there are a wide variety of obstacles that can prevent a learning algorithm from finding an acceptable solution.

The structure of the task can itself be a hindrance. The scheme that most algorithms are designed to tackle is that of a binary classification problem. This is characterized by the categorization of instances in one of two possible classes, which can be represented as a binary variable which acts as the target. Each instance is in turn represented by a vector valued in a set of variables. Although this is the simplest setting for a supervised machine learning problem, and many methods are initially created with it in mind for this reason, real world situations usually need more complex approaches.

One possible case is problem objects being represented by several data points, either homogeneous or heterogeneous according to whether they come from the same feature space. For example, one molecule may present various forms [12], where one of them may present a valid solution, rendering that molecule apt to solve the studied problem. This is known as a *multi-instance* learning task, while a *multi-view* one would have the data points belonging to different sources (and different feature spaces) [13], like social media posts with associated image and text. Learning methods are typically applied with a one-to-one input-target association in mind, so these types of input structures become harder to work around.

[12]: Dietterich et al. (1997), “Solving the multiple instance problem with axis-parallel rectangles”

[13]: Sun (2013), “A survey of multi-view machine learning”

Equivalently to each problem case being structured differently to a feature vector, the target can also become more complex. Many real world situations are better represented with more than one target variable. For example, tags in text documents can appear simultaneously, so predicting them will require a multilabel classification algorithm [14]. A similar case applies to continuous values in targets, leading to multi-output regression methods [15].

[14]: Herrera et al. (2016), "Multilabel classification"

[15]: Borchani et al. (2015), "A survey on multi-output regression"

Moreover, combinations of the previous nonstandard inputs and outputs structures can coexist, like in multi-instance multilabel problems [16] or multi-view multilabel ones [17].

[16]: Zhou et al. (2012), "Multi-instance multi-label learning"

[17]: Zhu et al. (2020), "Global and local multi-view multi-label learning"

Different issues can arise that make certain classes hard to learn by classifiers due to their size, location or lack of representation within the training set itself. This phenomenon is broadly known as data complexity [18], and in some cases as difficult classes [19], if the complications are related to a specific class. Data complexity can be caused by the feature set not fully being able to separate instances of different classes, class boundaries being highly nonlinear or composed of several disjoint subsets, or a class being notably underrepresented with respect to the rest, among other factors.

[18]: Ho et al. (2002), "Complexity measures of supervised classification problems"

[19]: Galar et al. (2014), "Empowering difficult classes with a similarity-based aggregation in multi-class classification problems"

2.3 Deep learning

An alternative approach to extracting features before training a predictor is to embed the feature extraction stage within the untrained model itself, and learn the best features at the same time that the final model (a classifier, regressor, segmentor...) is trained. When this process is organized layer-wise, the overall model is called a *deep learning* model [20].

[20]: Goodfellow et al. (2016), *Deep learning*

Deep learning model design comprises two complementary elements: the type of *layers* that build up the network and determine the type of data that can be processed, and the *architecture* itself, that is, the way these layers are organized and allow to perform one task or another with the provided data. For example, one could build a classification network out of recurrent units for sentiment analysis, or an encoder-decoder network out of convolutional layers for image segmentation. The fundamental types of layers as well as the most relevant architectures are explained next.

Neural networks according to layer operations

Not every kind of neural network is prepared to deal with every type of dataset. Their main advantage is that they can become specialized in certain structures, such as sequential data (sound, speech, language), bidimensional data (images) and three-dimensional data (video). This specialization while preserving the same training strategies and essential implementation methods is what sets them apart from traditional learning methods, which are more fixed in their way of working through data.

Dense networks

Dense deep networks, also known as fully connected networks, are essentially the same as MLPs. Their main operation in each layer is matrix product, where a parameter matrix is used to extract the values of each layer out of those of the previous one. More formally, is x is an input vector, W is the parameter matrix and b is a bias parameter, the dense layer performs the following computation:

$$f(x) = Wx + b \quad W \in \mathscr{M}_{n\times m}(\mathbb{R}), x \in \mathbb{R}^m, b \in \mathbb{R}^n \ . \qquad (2.1)$$

These networks are called fully connected because each value in the output is able to draw information from all values in the input vector. This is especially useful when variables are not structured since the order of variables will not affect the training.

Convolutional networks

Convolutional networks (CNNs) emerge out of the need to adapt operations to bidimensional data, as well as reduce the computational complexity of dense networks when treating this type of high-dimensional data. Since the matrix used in a dense layer has $n \times m$ parameters, m being the number of variables in the input vector and n the number of variables in the output vector, the amount of floating point operations required to compute the result is $O(nm)$.

In a CNN, a certain number of matrices typically named *channels* is computed out of the input matrix. Each one is composed of values calculated by convoluting the original matrix with a weight matrix, called *kernel*, which is usually of a fixed small size: 3×3 up to 9×9. Each pixel (i, j) in a filter resulting from convoluting a kernel K over the input I can be computed as follows:

$$f(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \ . \qquad (2.2)$$

By keeping k, the size of the kernels, notably smaller than the input image, the complexity of convolution is $O(kn)$ and it requires much fewer parameters than matrix multiplication. This makes

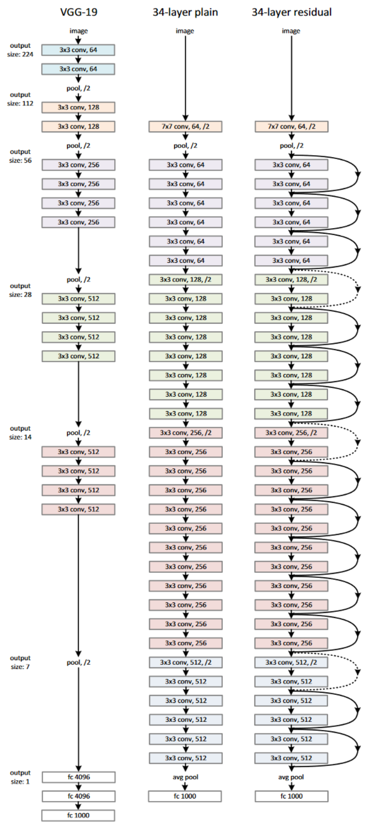


Figure 2.1: Comparison of the architectures of several CNNs, from left to right: VGG-19, a CNN with 36 layers and a residual CNN with 36 layers. Figure from [21].

CNNs appropriate for image-related tasks such as image classification, segmentation and object detection. They can also extend to tridimensional data such as video segments.

Most current neural network libraries implement cross correlation instead of the discrete convolution, which does not affect the results since equivalent kernels can be learned for it. The operation is still called convolution in most cases, and the only change is a sign flip for m and n:

$$f(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)\,. \tag{2.3}$$

Convolution is not the only stage performed by CNN. The other important step is *pooling*. This function outputs a smaller version of its input by summarizing nearby values. For example, max pooling [22] takes the maximum out of a rectangle of the input matrix and outputs that as a single value, reducing in this way the size of the matrix.

[22]: Zhou et al. (1988), "Computation of optical flow using a neural network."

Computer vision has been one of the fundamental applications of deep learning and CNNs have evolved greatly as a result, with many improvements and extensions such as residual connections (see Figure 2.1), breakdown of convolutions into smaller ones [23], and pruning strategies [24].

[23]: Szegedy et al. (2016), "Rethinking the Inception Architecture for Computer Vision"

[24]: Lin et al. (2019), "Towards optimal structured cnn pruning via generative adversarial learning"

Recurrent networks

In a recurrent neural network (RNN), some parts of the computation of the network at each step, e.g. the output, are fed as input in the next one. This creates a "memory" which allows the RNN to remember previous outputs when making predictions. RNNs are used for tasks such as speech recognition and language translation, where the order of the input is important and past outputs are relevant for the next predictions.

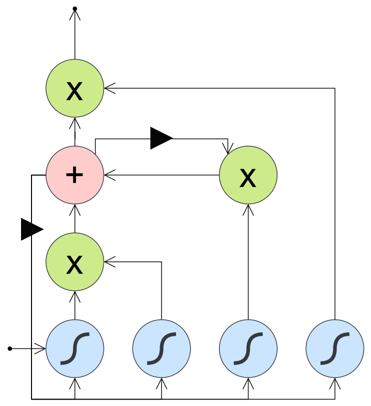


Figure 2.2: Diagram for an LSTM where blue units are gates (sigmoidal or tanh activations), green units are products and pink units are sums. Triangles over data flows indicate values that are fed at the next step.

A popular kind of RNN units are long short-term memory (LSTM) units [25], which add a self-loop controlled (*gated*) by another unit so that the memorized information can be eventually forgotten. See Figure 2.2 for a detailed schematic of this type of unit.

[25]: Hochreiter et al. (1997), "Long short-term memory"

Attention and transformers

Attention mechanisms started in encoder-decoder recurrent networks as a system to identify parts of speech that are more relevant than others within the input data while decoding is in process [26, 27]. Instead of compacting all the inputs into an encoded vector and decoding from there, attention allows to have all information available and focus on just the important pieces at each decoding step. There exist several ways of applying attention according to the elements that interact with one another, including self-attention, local attention and global attention.

[26]: Bahdanau et al. (2015), “Neural machine translation by jointly learning to align and translate”
[27]: Luong et al. (2015), “Effective approaches to attention-based neural machine translation”

Based on the concept of attention, transformers [28] were conceptualized by simply avoiding recurrent units and building the whole network around stacked self-attention operations. Figure 2.3 shows how a transformer is organized. Transformers have been applied beyond machine translation to many other natural language tasks, where they are currently the state of the art [29]. Some proposals have tackled computer vision as well [30], but are being matched in performance by attention-free models such as [31].

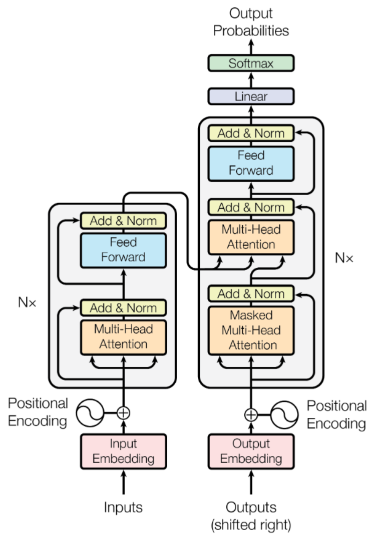


Figure 2.3: Original architecture diagram of the Transformer. Source: [28].

[28]: Vaswani et al. (2017), “Attention is all you need”
[29]: Devlin et al. (2019), “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”
[30]: Liu et al. (2021), “Swin transformer: Hierarchical vision transformer using shifted windows”
[31]: Chen et al. (2021), “CycleMLP: A MLP-like architecture for dense prediction”

Network architectures

Neural layers can be organized in different formations and connected in various ways in order to achieve specific solutions. The structure and connections of a network are known as its architecture.

Classifiers and regressors

Classifiers and regressors are very common types of neural network architectures. The key to obtaining a label output from a network is to stack layers which transform and progressively reduce the dimension of the original data, up to the last layer where the class is selected or a numerical value is predicted.

Some classification networks are trained and tested against well known benchmarks, becoming a reference for further works and even a basis for other tasks, leveraging the already extracted knowledge by means of *transfer learning*. For instance, the VGG-19 and ResNet networks shown in Figure 2.1 are standard tools for general image classification.

Encoder-decoder structures

There exists a category of deep architectures composed of two components, an *encoder* and a *decoder*, where there is an interest in the model operating first with the features in order to obtain higher-level features (encoding) and then developing these features back onto more detailed and specific versions.

For example, when the objective task is to segment the pixels in an image, that is, label each pixel with one of several classes, a possible solution is to compute abstract, high-level features for the image, and use those to classify each pixel next [32]. This allows to analyze the neighborhoods of each pixel before assigning it to a class, which will probably lead to more cohesive segmentations. Using an encoder-decoder structure, the encoder would compute these low-resolution but high-level features, and the decoder would perform the detailed labeling task out of the extracted information.

[32]: Minaee et al. (2021), "Image segmentation using deep learning: A survey"

Similarly, an encoder-decoder scheme made out of recurrent units, also known as a *sequence-to-sequence* model [33], could serve as basis for a language translation system. The encoder would extract an intermediate representation for the meaning of the original sentence and the decoder would transform that onto the target language.

[33]: Prabhavalkar et al. (2017), "A Comparison of Sequence-to-Sequence Models for Speech Recognition."

A special subset of encoder-decoder architectures are autoencoders, which are further described next.

Autoencoders

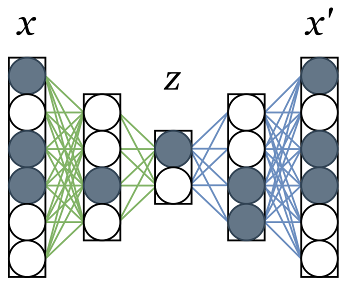


Figure 2.4: Schematic structure of a fully connected autoencoder.

Essentially, an *autoencoder* is an encoder-decoder architecture which is trained to map its inputs onto its outputs. Being f the encoder network and g the decoder one, the main objective of an autoencoder would be to match the input data as closely as possible (see Figure 2.4):

$$x \approx g(f(x)) \tag{2.4}$$

As a feature learner, the autoencoder trains to extract appropriate features from data by considering that quality features should allow to reconstruct the original data from the encoded vectors.

Different variations can be introduced into the training process and the autoencoder structure in order to extend its functionality. For example, the encoded features can be manipulated to be more sparse, i.e. most of them are equal to 0 for each input vector; the extracted reconstruction can discard noise from the inputs, or it can model the data as a continuous probability distribution. Article I goes into detail about these variants and compares autoencoders to other feature learning methods.

Autoencoders are not limited to the existing variants. Since they are usually based on a penalty function added to the objective that they train to optimize, an expert can define a different penalty function and induce a new behavior on the learned features. This penalty function, unlike the basic objective, can be based on network inputs, outputs, encodings or even external data associated to each instance. This idea is further explored in Article V, where three novel autoencoder models are proposed.

Unlike some types of neural networks which are used more commonly such as CNN classifiers or natural language models, autoencoders are not straightforward to implement in current deep learning libraries. This is due to autoencoders having two main output points where most neural networks have one: autoencoders can both encode a feature vector through their middle layer and reconstruct it through their last layer. In spite of this, there are few software tools which act as an abstraction layer over deep learning platforms, giving easier access to autoencoder functionalities. This, along with our developed software package Ruta, is further discussed in Article II.

Autoencoders, just like other kinds of networks, can adapt to process data with different structures. As long as the input and output *shapes* of the network coincide, an autoencoder will be able to transform the variables and produce an output as close as possible to the sampled data. In consequence, autoencoders can be made out of dense layers, convolutional layers and recurrent units as detailed in Section 3.1. Other possible structures for problems which could be captured by an adequately built autoencoders are analyzed in Article III.

Finally, the utility of autoencoders as feature learners is not limited to helping classification methods by projecting instances to a more useful feature space. As mentioned before, autoencoders may learn noise-resilient representations which allow to restore noisy signals. The reconstruction fidelity can also be used as a sign of anomalous data. Meanwhile, the encoded features can also be binarized to serve as bucket identifiers or treated as a probability distribution to sample new points. All of these applications are explained and demonstrated in Article IV.

References

[1] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[2] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[3] Rich Caruana and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms". In: *Proceedings of the 23rd International Conference on Machine Learning* (2006). DOI: 10.1145/1143844.1143865.

[4] M Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms*. Springer, 2016.

[5] Jesper E Van Engelen and Holger H Hoos. "A survey on semi-supervised learning". In: *Machine Learning* 109.2 (2020), pp. 373–440. DOI: 10.1007/s10994-019-05855-6.

[6] Jesper E. van Engelen and Holger H. Hoos. "A survey on semi-supervised learning". In: *Machine Learning* 109 (2019), pp. 373–440. DOI: 10.1007/s10994-019-05855-6.

[7] Francisco Herrera, Cristóbal José Carmona, Pedro González, and María José del Jesús. "An overview on subgroup discovery: foundations and applications". In: *Knowledge and Information Systems* 29 (2010), pp. 495–525. DOI: 10.1007/s10115-010-0356-2.

[8] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285. DOI: 10.1613/jair.301.

[9] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489. DOI: 10.1038/nature16961.

[10] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50.

[12] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. "Solving the multiple instance problem with axis-parallel rectangles". In: *Artificial Intelligence* 89.1-2 (1997), pp. 31–71. DOI: 10.1016/S0004-3702(96)00034-3.

[13] Shiliang Sun. "A survey of multi-view machine learning". In: *Neural Computing and Applications* 23.7 (2013), pp. 2031–2038. DOI: 10.1007/s00521-013-1362-6.

[14] Francisco Herrera, Francisco Charte, Antonio J Rivera, and María J del Jesus. "Multilabel classification". In: *Multilabel Classification*. Springer, 2016, pp. 17–31. DOI: 10.1007/978-3-319-41111-8_2.

[15] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larranaga. "A survey on multi-output regression". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 216–233. DOI: 10.1002/widm.1157.

[16] Zhi-Hua Zhou, Min-Ling Zhang, Sheng-Jun Huang, and Yu-Feng Li. "Multi-instance multi-label learning". In: *Artificial Intelligence* 176.1 (2012), pp. 2291–2320. DOI: 10.1109/TPAMI.2018.2861732.

[17] Changming Zhu et al. "Global and local multi-view multi-label learning". In: *Neurocomputing* 371 (2020), pp. 67–77. DOI: 10.1016/j.neucom.2019.09.009.

[18] Tin Kam Ho and Mitra Basu. "Complexity measures of supervised classification problems". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.3 (2002), pp. 289–300. DOI: 10.1109/34.990132.

[19] Mikel Galar, Alberto Fernández, Edurne Barrenechea, and Francisco Herrera. "Empowering difficult classes with a similarity-based aggregation in multi-class classification problems". In: *Information Sciences* 264 (2014), pp. 135–157. DOI: 10.1016/j.ins.2013.12.053.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[22] Yi-Tong Zhou and Rama Chellappa. "Computation of optical flow using a neural network." In: *IEEE 1988 International Conference on Neural Networks*. 1988, pp. 71–78. DOI: 10.1109/ICNN.1988.23914.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2818–2826. DOI: 10.1109/cvpr.2016.308.

[24] Shaohui Lin et al. "Towards optimal structured cnn pruning via generative adversarial learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 2790–2799. DOI: 10.1109/CVPR.2019.00290.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[26] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *Proceedings of the ICLR 2015*. 2015.

[27] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015. DOI: 10.18653/v1/D15-1166.

[28] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017).

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[30] Ze Liu et al. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 10012–10022. DOI: 10.1109/ICCV48922.2021.00986.

[31] Shoufa Chen, Enze Xie, Chongjian Ge, Ding Liang, and Ping Luo. "CycleMLP: A MLP-like architecture for dense prediction". In: *arXiv preprint arXiv:2107.10224* (2021).

[32] Shervin Minaee et al. “Image segmentation using deep learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: 10.1109/TPAMI.2021.3059968.

[33] Rohit Prabhavalkar et al. “A Comparison of Sequence-to-Sequence Models for Speech Recognition.” In: *Interspeech*. 2017, pp. 939–943. DOI: 10.21437/Interspeech.2017-233.

Technical details 3

Bringing all previous theoretical concepts into a working model is not straightforward if approached from scratch but, since the late 2000s and early 2010s, there have been notable advancements in programming libraries which take care of the hard work of calculating gradients and optimizing weights, leaving to the data scientist just the task of designing an appropriate model.

This chapter is dedicated to illustrating the reader on the different possibilities that exist for designing and implementing autoencoder networks. Detailed examples on how to code simple autoencoders are provided.

3.1 Design

Designing an autoencoder for a certain task can be challenging, since the objective is to find a more useful representation of the data but we cannot know the size of the optimal representation beforehand, thus difficulting decisions about the number of layers and the size of each one.

Type of layers

As explained in greater detail in Section 2.3, different layers are available in every deep learning framework and can be used according to the structure of the provided data and the kind of operations the practitioner wants to apply to it. The choice in an autoencoder would be analogous to that in other kinds of networks:

- **Unstructured variables**: the most basic type of data takes the form of tabular values where columns may be related but do not hold a specific structure, they can be reordered in any way and the data is still valid. Appropriate layers for this kind of data are dense layers, also known as fully connected or linear. These compute a product between the input vector and a matrix of parameters, which gives a new output vector as a result.

- **Images**: the recent surge in deep learning applications is in great part due to the potential of convolutional operations for image processing. These relieve a great part of the computational complexity of dense layers by leveraging the spatial relations among values. Convolutional layers are frequently coupled with pooling layers (either max-pooling or average-pooling) which reduce the dimensionality of the data points, as well as dropout layers which randomly disable some layer nodes during training in order to improve model robustness.
- **Sound, time series and sequential data**: many data sources impose a one-dimensional structure to the values, e.g. recorded sounds, stock prices, sensor signals across time, etc. Convolutional operations can also apply in this case, since they can operate in one dimension analogously to the two-dimensional version. However, other layers have been specifically designed for this kind of data, such as long short-term memory (LSTM) units or gated recurrent units (GRU). These are encompassed under the term *recurrent units*.

Model depth

Determining how deep a model should become, i.e., how many layers to stack, is a process influenced by the amount of variables and instances in the dataset. Taking into account that deep learning models are usually data-hungry, defining a model with many layers will require a large dataset in order to optimize all parameters. This inconvenience is especially present in the case of dense layer-based models, since these have many more parameters than convolutional models.

Encoding layer

Autoencoders being mainly feature learners, the most important layer is that where the new representation of the data will be extracted. Some aspects that are important to evaluate are the dimension or shape of the encoding and the activation function.

The dimension of the encoding layer will determine the compactness of the new representation. Thus, if the objective is to find a small set of variables for a dataset, a short length will be selected for this layer. The optimal size can be hard to find, but if the layer is too small the behavior of the autoencoder will generally be poor (the loss function will not decrease during training) so a practical way of estimating an appropriate size is to start with a very short layer and increase the size until the training process is able to converge.

For its part, the activation function will determine the range of the values that will be obtained. This is relevant if, for instance, bounded values are needed for a later purpose. In that case, a sigmoid or a tanh activation function would be helpful in providing values within the $[0, 1]$ and $[-1, 1]$ ranges, respectively.

Output layer

Finally, the output layer needs to preserve the same structure of the input data. This means that, if an instance is composed of n values, this layer needs to produce n outputs.

It is also convenient to consider the range of the input values if an activation is to be applied to the final layer. Using the appropriate activation function in the output layer can facilitate the reconstruction task of the autoencoder.

- **Unbounded**: for data with unbounded variables, no activation (also known as "linear" activation in some frameworks) is the adequate decision, since most activations restrict the range of the output.
- **Partially bounded**: Nonnegative data can be generated by a ReLU activation.
- **Bounded interval**: Data in the $[0, 1]$ range can be produced as the result of a sigmoid activation. Similarly, the tanh activation function can provide data in the $[-1, 1]$ range.

Architecture search

Even if the previous guidelines can provide a starting point for the design of an appropriate autoencoder for a certain task, many hyperparameters remain for the user to set: number of total layers, number of neurons in each layer, loss function penalties to add, etc. The search for an optimal model can become difficult if performed by trial and error, even for experienced practitioners. There are some solutions in order to facilitate this process. An extensive experimentation serving to identify which autoencoder variants and structures are more suitable to tackle different types of problems is available at [1]. The alternative is running an automatic search for an adequate architecture under some heuristic, for example, an evolutionary method [2]. This option falls under the category of algorithms known as neural architecture search [3].

[1]: Pulgar et al. (2020), “Choosing the proper autoencoder for feature fusion based on data complexity and classifiers: Analysis, tips and guidelines”

[2]: Charte et al. (2020), “EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search”

[3]: Elsken et al. (2019), “Neural Architecture Search: A Survey”

3.2 Training deep models

The training process of a deep learning model can be very resource-intensive, since it requires computing hundreds of arithmetic operations, usually across several thousands of parameters.

Stochastic gradient descent

Deep learning models are usually optimized by differentiating the objective function with respect to the model parameters and updating those in the direction of steepest descent. This optimization scheme is known as *gradient descent*, but it suffers from high time complexity when datasets become large. An approximation called *stochastic gradient descent* (SGD) is used instead as a result [4]. It simply runs gradient descent on successive minibatches of data, estimating the true gradient as the overall expectation of all gradients computed. This fixes the time required for each parameter update, depending only on the size of the minibatch and independently from the training set size.

[4]: Goodfellow et al. (2016), *Deep learning*

Gradient descent is prone to converging to local minima and SGD is just an approximation which will also tend to stabilize in local minima or locations with small gradients. However, neural networks do not represent convex functions, on the contrary, they almost always present a large amount of local minima. As a consequence, many improvements have been proposed to find better solutions with lower values of the objective function. Of course, they still do not ensure that a global minimum is reached, but they are more likely to find an overall well optimized point in parameter space. Some of these improvements on SGD are AdaBoost [5], Adadelta [6] and Adam [7].

[5]: Freund et al. (1997), "A decision-theoretic generalization of on-line learning and an application to boosting"

[6]: Zeiler (2012), "Adadelta: an adaptive learning rate method"

[7]: Kingma et al. (2014), "Adam: A method for stochastic optimization"

Gradient computation

In order to optimize a neural network, SGD requires to compute the value of the objective function and then find its gradient with respect to model parameters. Since SGD is an approximation, the training stage is usually divided into a series of epochs, during which a batch of data is fed to the network. Each epoch is composed of several forward and backward passes.

Forward pass

The value of the objective function is the result of transforming the input data through the network up to the application of the loss function, that is, the metric that evaluates the obtained output with respect to the desired one. This sequence of operations is known as a *forward pass* of a batch of data.

Backward pass

The expression for the derivative of each layer in a feedforward neural network turns out to be dependent on the values for the derivatives in the next layer. The strategy, as a result, is to perform the computations starting from the output layer back to the input layer. This technique is known as *backpropagation of errors*.

The derivatives are computed for the last layer and, using the chain rule, those values can be reused to find the derivatives for the second-to-last layer, then the layer before, and the rest successively. This backward pass is what allows the optimizer to calculate the gradient of the objective function with respect to all the weights.

After one or more backward passes, the errors and derivatives obtained are combined and used to update the weights in the direction of steepest descent. Certain properties of the optimizer, such as momentum, may also be applied in order to compensate for potential noise and side effects of using only a small part of the dataset for each update.

3.3 Implementation

During the latest years, there has been a notable evolution in the scene of software libraries for deep learning. From the existence of a wide variety of them with differing functionalities, ease of use and optimizations, there has been a tendency to condensate popularity in just two of them, which currently offer very similar functionalities and interfaces: Tensorflow and Pytorch.

About Tensorflow

Tensorflow is a deep learning framework implemented in C++, with Python and C++ interfaces, developed by Google and open source contributors.

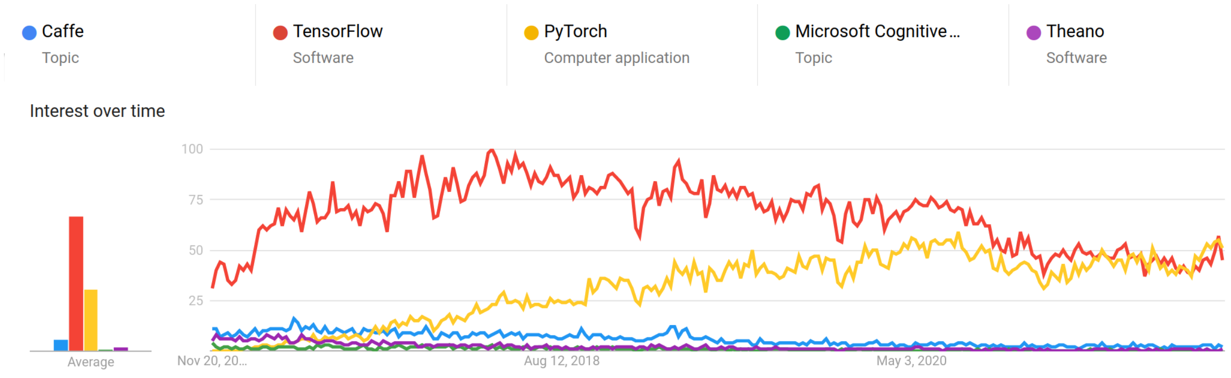


Figure 3.1: Trends for web searches for five of the most popular deep learning frameworks, over the last 5 years.

This library provides both lazy and eager execution of tensor operations on either CPU or GPU. Eager execution, the usual way of running statements where the result of each statement is readily available just after an operation is run, is enabled by default starting from Tensorflow 2. Lazy execution behaves the opposite, delaying the actual computation of operations until a final step where everything is processed jointly. It allows to optimize models better using a computation graph, but makes it harder to debug them.

[8]: Chollet et al. (2022), *Keras API Documentation*

Tensorflow integrates an easy-to-use API called Keras [8], which raises the level of abstraction so that the user does not need to program each operation but can design a network based on its layers. This API also brings additional tools including prebuilt and pretrained models, data manipulation functionalities, built-in datasets and automatic parameter tuning.

Sample implementation

Consider an essential autoencoder model where both the encoder and the decoder are composed of one fully connected layer.

For the purpose of modularity, which helps reusing parts of the code later, it is convenient to define the encoder and the decoder separately. Each will be represented by an object of class `Sequential`, and comprises a list with just one layer, of class `Dense`.

[7]: Kingma et al. (2014), "Adam: A method for stochastic optimization"

Both models are chained by listing them in a new `Sequential` model, which is then compiled to optimize the binary crossentropy loss using the Adam [7] optimizer. Assuming that the variable `x_train` holds the training data, the model is optimized when the `fit()` method is called.

Once the autoencoder has been trained, the encoder can be fed new instances from the same variable space and it will map those to the learned representation.

```python
import tensorflow as tf

enc_dim = 10
encoder = tf.keras.Sequential([
    tf.keras.layers.Dense(enc_dim, activation="relu",
        input_shape=(x_train.shape[1], ))
])
decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(x_train.shape[1],
        activation="sigmoid", input_shape=(enc_dim,))
])

autoencoder = tf.keras.Sequential([encoder, decoder])
autoencoder.compile(loss="binary_crossentropy",
    optimizer="adam")
autoencoder.fit(x_train, x_train, epochs=10)

new_encodings = encoder.predict(x_test)
```

About Pytorch

Pytorch is the successor of the Lua-based Torch library, with the objective of providing an interface as natural as possible for the experienced Python user. It is developed by Facebook and other open source contributors.

This library attempts to provide a deeper integration with Numpy and Scipy, and is designed to work in eager execution but models can be transformed into graph mode (which enables lazy execution and further optimizations).

The Pytorch API is well organized into modules depending on the type of data that is being processed: `torchvision` for images, `torchaudio` for audio sequences and `torchtext` for text and natural language. Models can be built in a similar fashion to the Keras interface, but the training process usually requires more explicit code.

Sample implementation

Next is an equivalent implementation for the same autoencoder of the previous example, this time in Pytorch. The similarities can be appreciated when defining the encoder and the decoder, but the optimization of the autoencoder model is done more explicitly this time, by means of a training loop. At each iteration, the following steps are performed:

1. A batch of input samples is selected
2. The neural network is fed these samples and its output is obtained
3. A loss metric is calculated according to the target outputs
4. The gradients applied to the model are reset
5. The gradients corresponding to the current loss are computed and propagated
6. Model parameters are updated by the optimizer according to the gradients

Once the training loop has finished, the model is considered trained and is switched to evaluation mode in order to allow for extraction of the learned features for new data.

```python
import torch
from torch.nn.functional import binary_cross_entropy

enc_dim = 10

encoder = torch.nn.Sequential(
    torch.nn.Linear(x_train.shape[1], enc_dim),
    torch.nn.ReLU(True)
)
decoder = torch.nn.Sequential(
    torch.nn.Linear(enc_dim, x_train.shape[1]),
    torch.nn.ReLU(True)
)

autoencoder = torch.nn.Sequential(encoder, decoder)

optimizer = torch.optim.Adam(autoencoder.parameters())

for i in range(100):
    inputs = x_train[i*10:(i+1)*10]
    output = autoencoder(inputs)
    loss = binary_cross_entropy(output, inputs)
    autoencoder.zero_grad()
    loss.backward()
    optimizer.step()

autoencoder.eval()
encoder(x_test)
```

GPU parallelization

From a hardware point of view, neural network training is a relatively easy task since it mostly involves great amounts of floating point matrix products. Thus, it does not require the complexity of a whole CPU to run the optimization task. Instead, most of the bulk of the process can be carried out by the simpler, more dense computing chips that are present in current graphical processing units (GPU). These allow very fast parallel calculations and, as a result, are ideal to accelerate the training process.

Most tensor libraries, including Tensorflow and Pytorch, are implemented with GPU execution in mind as well as CPU execution (usually as fallback) of the models. The most popular platform for parallel execution is CUDA from NVIDIA [9], which provides access to an instruction set that is executed on a GPU, and OpenCL [10] is an equivalent standard. If the correct drivers and the CUDA platform are installed, the selected library will be able to automatically connect to the GPU in order to copy data to the dedicated memory and send instructions to carry out the necessary computations.

[9]: NVIDIA (2022), *CUDA Toolkit Documentation*

[10]: Khronos Group (2022), *OpenCL Resource Guide*

The potential of GPUs is further leveraged in dedicated multi-GPU servers where the calculations can be spread across several accelerators, resulting in proportional time savings with no loss in quality of results. Graphics cards manufacturers currently produce variants of the cards that are specific for computing purposes and discard the video output, sometimes including dedicated circuitry for tensor operations (e.g. Tensor Cores in the NVIDIA A100 lineup [11] as can be seen in Figure 3.2).

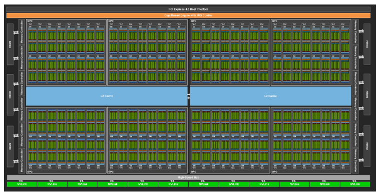


Figure 3.2: NVIDIA A100 GPU including 108 Tensor cores. Source: [11].

[11]: NVIDIA (2020), *NVIDIA A100 Tensor Core GPU Architecture*

References

[1] Francisco J Pulgar, Francisco Charte, Antonio J Rivera, and Maria J del Jesus. "Choosing the proper autoencoder for feature fusion based on data complexity and classifiers: Analysis, tips and guidelines". In: *Information Fusion* 54 (2020), pp. 44–60. DOI: 10.1016/j.inffus.2019.07.004.

[2] Francisco Charte, Antonio J Rivera, Francisco Martinez, and Maria J del Jesus. "EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search". In: *Integrated Computer-Aided Engineering* 27.3 (2020), pp. 211–231. DOI: 10.3233/ICA-200619.

[3] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[5] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. DOI: 10.1006/jcss.1997.1504.

[6] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[7] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[8] François Chollet and contributors. *Keras API Documentation*. https://keras.io/api/. 2022.

[9] NVIDIA. *CUDA Toolkit Documentation*. https://docs.nvidia.com/cuda/. 2022.

[10] Khronos Group. *OpenCL Resource Guide*. https://www.khronos.org/api/opencl/resources. 2022.

[11] NVIDIA. *NVIDIA A100 Tensor Core GPU Architecture*. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. Accessed: 2022-04-26. 2020.

Published articles

Summary of published results 4

This part of the document reproduces five articles published in peer-reviewed journals authored by the candidate. They represent the core contributions of the present thesis. However, there are many more outcomes resulting from the research work produced during the course of the doctoral studies. These include other journal articles co-authored in collaboration with other colleagues in tangential topics, national and international conferences, disseminative talks in seminars, several software libraries and one book. Figure 4.1 indicates the main numbers related to the different types of works developed.

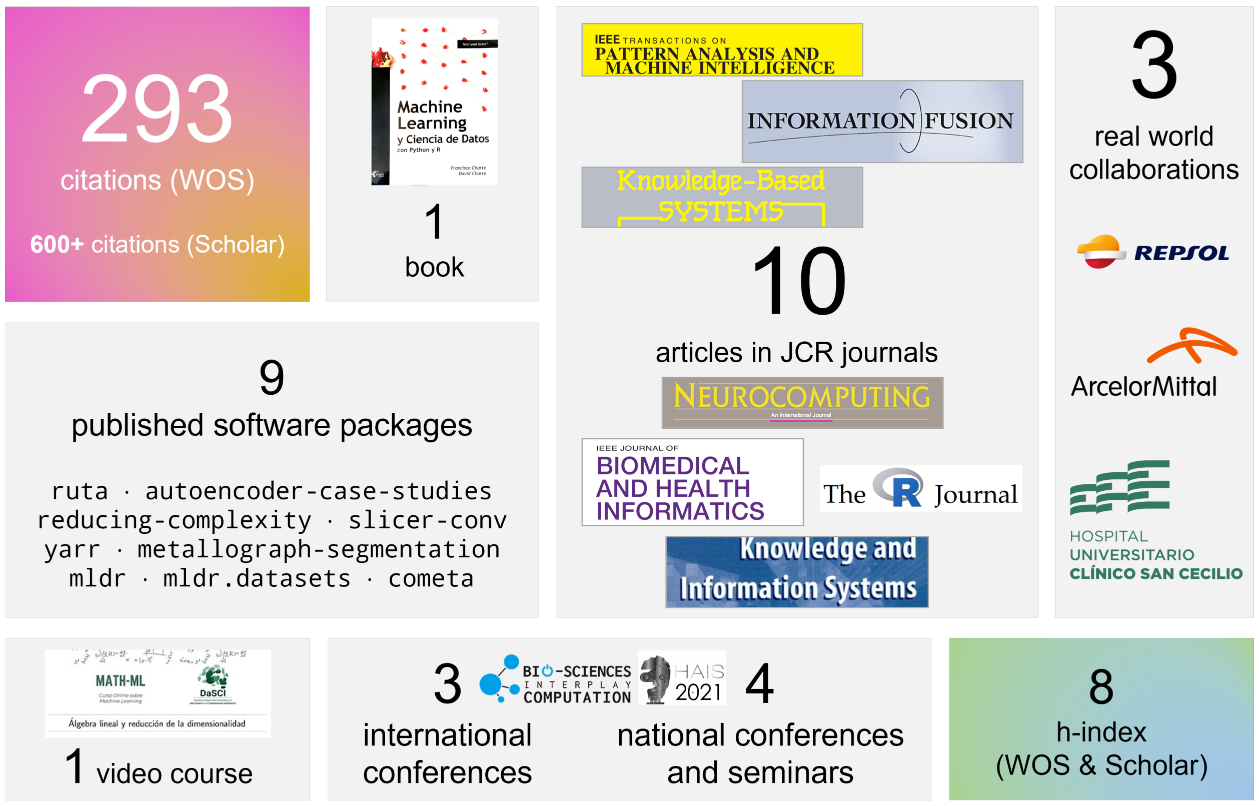

293
citations (WOS)
600+ citations (Scholar)
Machine Learning y Ciencia de Datos con Python y R
1
book
IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE
INFORMATION FUSION
Knowledge-Based SYSTEMS
10
articles in JCR journals
NEUROCOMPUTING
An International Journal
IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS
The R Journal
Knowledge and Information Systems
3
real world collaborations
REPSOL
ArcelorMittal
HOSPITAL UNIVERSITARIO CLÍNICO SAN CECILIO
9
published software packages
ruta · autoencoder-case-studies
reducing-complexity · slicer-conv
yarr · metallograph-segmentation
mldr · mldr.datasets · cometa
MATH-ML
DaSCI
Álgebra lineal y reducción de la dimensionalidad
1 video course
3
BIO-SCIENCES INTERPLAY COMPUTATION
HAIS 2021
4
international conferences
national conferences and seminars
8
h-index
(WOS & Scholar)

Figure 4.1: Visual summary of the main results obtained during the candidate's research career.

A quick look at the infographic reveals that the focus of this thesis has not only been to produce novel scientific research, but also useful software tools to apply and improve our developments, as well as didactic material which brings this area of computer science closer to different audiences.

In order to not only measure the work by its quantity but also by its impact, mainly within the research community, the joint number of citations through time is shown in Figure 4.2 as well as its position relative to the rest of publications in the same field in Figure 4.3.

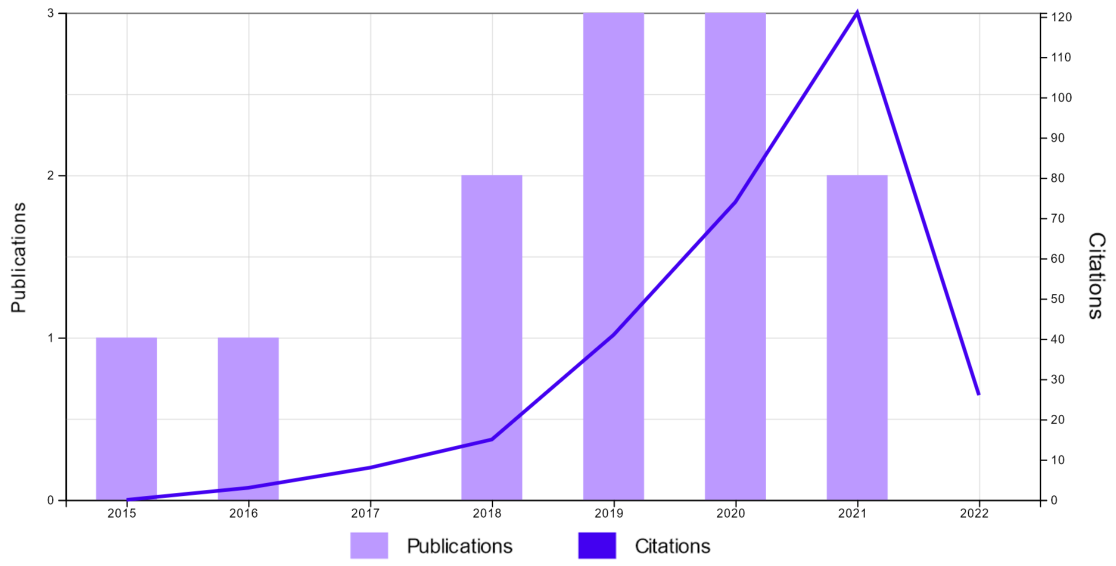


Figure 4.2: Graph of the number of publications and citations across years (source: Web of Science).

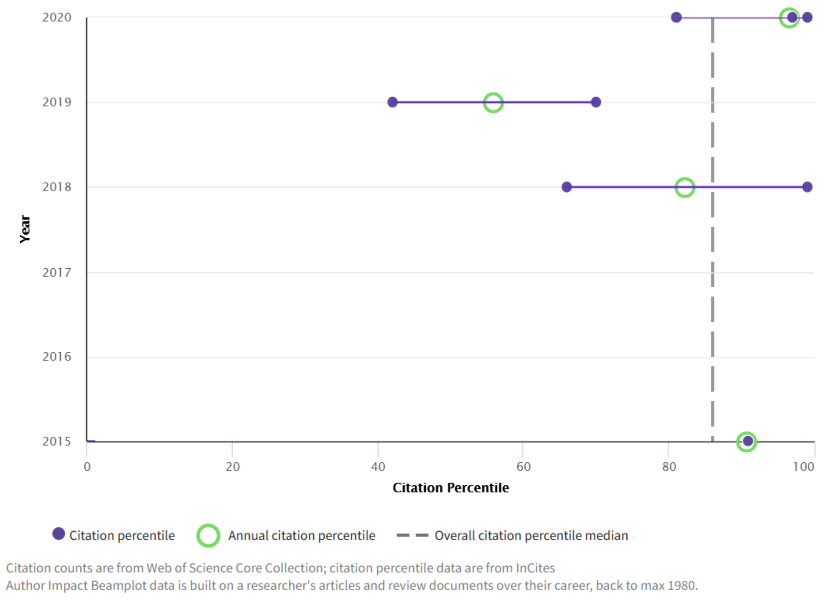


Figure 4.3: Beamplot displaying the impact of the candidate's publications since 2015 to 2020 (source: Web of Science). Each purple point corresponds to a publication and the horizontal position describes its citation level with respect to the rest of publications in the same area and year (higher is better).

4.1 Relation of all published material

The following tables detail all of the candidate's publications as of the time of writing. First, Table 4.1 indicates articles published in journals ranked within Journal Citation Reports (JCR), including quality metrics such as journal position in the ranking, quartile and number of citations.

The rest of publications such as articles in other journals, conference works, talks and one book are broken down in Table 4.2.

Table 4.1: Quality metrics of the articles published in JCR journals during the research period of the candidate. The position of each journal is taken from the JCR of the corresponding year, except for those marked with an asterisk (∗) which are taken from JCR 2020. The source for the number of citations is Web of Science. Category legends: Statistics and Probability (STAT), Computer Science/Artificial Intelligence (CS AI), Computer Science/Theory and Methods (CS TM), Computer Science/Information Systems (CS IS). Articles marked with a star (★) are part of the core of this thesis.

	Year	**Title**	**Journal**	**Position**	**Cat.**	**Cit.**	**Ref.**
	2015	Working with Multilabel Datasets in R: The mldr Package	The R Journal	56/123 (Q2)	STAT	30	[1]
	2018	Tips, guidelines and tools for managing multi-label datasets: The mldr.datasets R package and the Cometa data repository	Neurocomputing	28/134 (Q1)	CS AI	12	[2]
★	2018	A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines	Information Fusion	2/105 (Q1) 3/134 (Q1)	CS TM CS AI	102	[3]
★	2019	Ruta: Implementations of neural autoencoders in R	Knowledge-based Systems	15/137 (Q1)	CS AI	4	[4]
★	2020	An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges	Neurocomputing	30/139 (Q1)	CS AI	11	[5]
	2020	Artificial intelligence within the interplay between natural and artificial computation: Advances in data science, trends and applications	Neurocomputing	30/139 (Q1)	CS AI	35	[6]
	2020	COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images	Biomedical And Health Informatics	28/161 (Q1)	CS IS	55	[7]
	2021	Revisiting data complexity metrics based on morphology for overlap and imbalance: snapshot, new overlap number of balls metrics and singular problems prospect	Knowledge and Information Systems	65/139 (Q2)∗	CS AI	1	[8]
★	2021	Reducing Data Complexity using Autoencoders with Class-informed Loss Functions	Pattern Analysis and Machine Intelligence	1/139 (Q1)∗	CS AI	n/a	[9]
	2022	A tutorial on the segmentation of metallographic images: Taxonomy, new MetalDAM dataset, deep learning-based ensemble model, experimental analysis and challenges	Information Fusion	1/110 (Q1)∗ 3/139 (Q1)∗	CS TM CS AI	0	[10]

Table 4.2: Other publications, conferences and talks authored by the candidate. Articles marked with a star (★) are part of the core of this thesis.

	Year	Title	Type	Published in	Ref.
	2015	mldr: Paquete R para exploración de datos multietiqueta	National conference	XVI Conferencia de la Asociación Española para la Inteligencia Artificial	[11]
	2016	Análisis visual de técnicas de *deep learning* no supervisado	National conference	XVII Conferencia de la Asociación Española para la Inteligencia Artificial	[12]
	2016	R Ultimate Multilabel Dataset Repository	International conference	Hybrid Artificial Intelligence Systems	[13]
	2017	Unsupervised Deep Learning in R with Ruta	Poster	IX Jornadas de usuarios de R	
	2018	A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines	Keywork in national conference	XVIII Conferencia de la Asociación Española para la Inteligencia Artificial	[14]
★	2018	A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations	Journal article	Progress in Artificial Intelligence (115/171, Q3 in ESCI ranking; Q2 in SJR)	[15]
	2019	A showcase of the use of autoencoders in feature learning applications	International conference	International Work-Conference on the Interplay between Natural and Artificial Computation	[16]
	2019	Aplicaciones prácticas de las redes neuronales no supervisadas	National conference	Congreso esLibre 2019	
	2020	Autoencoders: An Overview and Applications	Talk	Severo Ochoa School on Machine Learning, Big Data, and Deep Learning in Astronomy (IAA-CSIC SOMACHINE)	
	2021	Machine Learning y Ciencia de Datos con Python y R	Book	Krasis Press	[17]
	2021	Slicer: Feature Learning for Class Separability with Least-Squares Support Vector Machine Loss and COVID-19 Chest X-Ray Case Study	International conference	Hybrid Artificial Intelligence Systems	[18]

4.2 Main articles

From the works previously mentioned, five peer-review articles have been selected as the main core of this thesis for advancing our fundamental objectives.

The first objective, as described previously in Section 1.4, was to study autoencoders and contribute to more people being able to use them to solve their tasks. Article I [3] addresses this by describing autoencoder fundamentals, its variants, implementations and providing tips on autoencoder design depending on the problem at hand.

[3]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

Continuing with the objective of facilitating access to autoencoders, Article II [4] presents Ruta, a software library which simplifies the creation and training process. It is programmed in the R language to ease the transition from other tools for users with little programming knowledge.

[4]: Charte et al. (2019), "Ruta: Implementations of neural autoencoders in R"

Next, part of the time was devoted to exploring a diverse range of problems where autoencoders could be applicable. Article III [15] organizes the knowledge around a variety of supervised problems, for example, multi-instance classification and multi-output regression. For its part, Article IV [5] analyzes another list of problems, mostly unsupervised, where autoencoder-based models have already been applied as a valid solution, from anomaly detection to instance generation.

[15]: Charte et al. (2019), "A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations"

[5]: Charte et al. (2020), "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges"

Some preliminary work was accomplished into approaching one of the nonstandard supervised problems, label distribution learning, using an autoencoder-based solution. This did not improve on current, simpler techniques so our focus changed onto tackling a more general situation that could be then applied to multiple supervised problems. This new development, consisting of a model which is able to extract features with better class separability than most traditional feature learners, is detailed in Article V [9]. We also demonstrate the usefulness of this novel solution by applying it to a specific problem, detection of COVID-related pulmonar symptoms in X-ray images, in [18].

[9]: Charte et al. (2021), "Reducing Data Complexity using Autoencoders with Class-informed Loss Functions"

[18]: Charte et al. (2021), "Slicer: Feature Learning for Class Separability with Least-Squares Support Vector Machine Loss and COVID-19 Chest X-Ray Case Study"

References

[1] Francisco Charte and David Charte. “Working with Multilabel Datasets in R: The mldr Package”. In: *R Journal* 7.2 (Dec. 2015), pp. 149–162. DOI: 10.32614/RJ-2015-027.

[2] Francisco Charte, Antonio J. Rivera, David Charte, Mara J. del Jesus, and Francisco Herrera. “Tips, guidelines and tools for managing multi-label datasets: The mldr.datasets R package and the Cometa data repository”. In: *Neurocomputing* 289 (May 2018), pp. 68–85. DOI: 10.1016/j.neucom.2018.02.011.

[3] David Charte, Francisco Charte, Salvador García, María José del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007.

[4] David Charte, Francisco Herrera, and Francisco Charte. “Ruta: Implementations of neural autoencoders in R”. In: *Knowledge-Based Systems* 174 (2019), pp. 4–8. DOI: 10.1016/j.knosys.2019.01.014.

[5] David Charte, Francisco Charte, Maria J del Jesus, and Francisco Herrera. “An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges”. In: *Neurocomputing* 404 (2020), pp. 93–107. DOI: 10.1016/j.neucom.2020.04.057.

[6] Juan M. Gorriz et al. “Artificial intelligence within the interplay between natural and artificial computation: Advances in data science, trends and applications”. In: *Neurocomputing* 410 (Oct. 2020), pp. 237–270. DOI: 10.1016/j.neucom.2020.05.078.

[7] S. Tabik et al. “COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images”. In: *IEEE Journal of Biomedical and Health Informatics* 24.12 (Dec. 2020), pp. 3595–3605. DOI: 10.1109/jbhi.2020.3037127.

[8] Jose Daniel Pascual-Triana, David Charte, Marta Andres Arroyo, Alberto Fernandez, and Francisco Herrera. “Revisiting data complexity metrics based on morphology for overlap and imbalance: snapshot, new overlap number of balls metrics and singular problems prospect”. In: *Knowledge and Information Systems* 63.7 (July 2021), pp. 1961–1989. DOI: 10.1007/s10115-021-01577-1.

[9] David Charte, Francisco Charte, and Francisco Herrera. “Reducing Data Complexity using Autoencoders with Class-informed Loss Functions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: 10.1109/tpami.2021.3127698.

[10] Julian Luengo et al. “A tutorial on the segmentation of metallographic images: Taxonomy, new MetalDAM dataset, deep learning-based ensemble model, experimental analysis and challenges”. In: *Information Fusion* 78 (Feb. 2022), pp. 232–253. DOI: 10.1016/j.inffus.2021.09.018.

[11] David Charte and Francisco Charte. “mldr: Paquete R para exploración de datos multietiqueta”. In: *XVI Conferencia de la Agencia Española para la Inteligencia Artificial*. Aepia. 2015.

[12] David Charte, Francisco Charte, and Francisco Herrera. “Análisis visual de técnicas de deep learning no supervisado”. In: *XVII Conferencia de la Agencia Española para la Inteligencia Artificial*. Aepia. 2016.

[13] Francisco Charte, David Charte, Antonio Rivera, María José del Jesus, and Francisco Herrera. “R ultimate multilabel dataset repository”. In: *International conference on hybrid artificial intelligence systems*. Springer. 2016, pp. 487–499. DOI: 10.1007/978-3-319-32034-2_41.

[14] David Charte, Francisco Charte, Salvador García, María José del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *XVIII Conferencia de la Agencia Española para la Inteligencia Artificial*. Aepia. 2018.

[15] David Charte, Francisco Charte, Salvador García, and Francisco Herrera. “A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations”. In: *Progress in Artificial Intelligence* 8.1 (2019), pp. 1–14. DOI: 10.1007/s13748-018-00167-7.

[16] David Charte, Francisco Charte, María José del Jesus, and Francisco Herrera. “A showcase of the use of autoencoders in feature learning applications”. In: *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer. 2019, pp. 412–421. DOI: 10.1007/978-3-030-19651-6_40.

[17] Francisco Charte and David Charte. *Machine Learning y Ciencia de Datos con Python y R*. Krasis Press.

[18] David Charte et al. “Slicer: Feature Learning for Class Separability with Least-Squares Support Vector Machine Loss and COVID-19 Chest X-Ray Case Study”. In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2021, pp. 305–315. DOI: 10.1007/978-3-030-86271-8_26.

A practical tutorial on autoencoders for nonlinear feature fusion

I

Source

David Charte, Francisco Charte, Salvador García, María José del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007

Abstract

Many of the existing machine learning algorithms, both supervised and unsupervised, depend on the quality of the input characteristics to generate a good model. The amount of these variables is also important, since performance tends to decline as the input dimensionality increases, hence the interest in using feature fusion techniques, able to produce feature sets that are more compact and higher level. A plethora of procedures to fuse original variables for producing new ones has been developed in the past decades. The most basic ones use linear combinations of the original variables, such as PCA (*Principal Component Analysis*) and LDA (*Linear Discriminant Analysis*), while others find manifold embeddings of lower dimensionality based on non-linear combinations, such as Isomap or LLE (*Linear Locally Embedding*) techniques.

More recently, autoencoders (AEs) have emerged as an alternative to manifold learning for conducting nonlinear feature fusion. Dozens of AE models have been proposed lately, each with its own specific traits. Although many of them can be used to generate reduced feature sets through the fusion of the original ones, there also AEs designed with other applications in mind.

The goal of this paper is to provide the reader with a broad view of what an AE is, how they are used for feature fusion, a taxonomy gathering a broad range of models, and how they relate to other classical techniques. In addition, a set of didactic guidelines on how to choose the proper AE for a given task is supplied, together with a discussion of the software tools available. Finally, two case studies

illustrate the usage of AEs with datasets of handwritten digits and breast cancer.

Keywords

autoencoders - feature fusion - feature extraction - representation learning - deep learning - machine learning

I.1 Introduction

The development of the first machine learning techniques dates back to the middle of the 20th century, supported mainly by previously established statistical methods. By then, early research on how to emulate the functioning of the human brain through a machine was underway. McCulloch and Pitts cell [2] was proposed back in 1943, and the Hebb rule [3] that the Perceptron [4] is founded on was stated in 1949. Therefore, it is not surprising that artificial neural networks (ANNs), especially since the backpropagation algorithm was rediscovered in 1986 by Rumelhart, Hinton and Willians [5], have become one of the essential models.

[2]: McCulloch et al. (1943), "A logical calculus of the ideas immanent in nervous activity"

[3]: Hebb (1949), *The organization of behavior: A neuropsychological theory*

[4]: Rosenblatt (1957), *The perceptron, a perceiving and recognizing automaton (Project PARA)*

[5]: Rumelhart et al. (1986), "Learning representations by back-propagating errors"

ANNs have been applied to several machine learning tasks, mostly following a supervised approach. As was mathematically demonstrated [6] in 1989, a multilayer feedforward ANN (MLP) is an universal approximator, hence their usefulness in classification and regression problems. However, a proper algorithm able to train an MLP with several hidden layers was not available, due to the vanishing gradient [7] problem. The gradient descent algorithm, firstly used for convolutional neural networks [8] and later for unsupervised learning [9], was one of the foundations of modern deep learning [10] methods.

[6]: Hornik et al. (1989), "Multilayer feedforward networks are universal approximators"

[7]: Hochreiter (1998), "The vanishing gradient problem during learning recurrent neural nets and problem solutions"

[8]: LeCun et al. (1989), "Backpropagation applied to handwritten zip code recognition"

[9]: Hinton et al. (2006), "A fast learning algorithm for deep belief nets"

[10]: Jürgen Schmidhuber (2015), "Deep Learning in Neural Networks: An Overview"

Under the umbrella of deep learning, multiple techniques have emerged and evolved. These include DBNs (*Deep Belief Networks*) [11], CNNs (*Convolutional Neural Networks*) [12], RNNs (*Recurrent Neural Networks*) [13] as well as LSTMs (*Long Short-Term Memory*) [14] or AEs (*autoencoders*).

[11]: Hinton (2009), "Deep belief networks"

[12]: LeCun et al. (1998), "The Handbook of Brain Theory and Neural Networks"

[13]: Williams et al. (1989), "A learning algorithm for continually running fully recurrent neural networks"

[14]: Hochreiter et al. (1997), "Long Short-Term Memory"

The most common architecture in unsupervised deep learning is that of the *encoder-decoder* [15]. Some techniques lack the encoder or the decoder and have to compute costly optimization algorithms to find a code or sampling methods to reach a reconstruction, respectively. Unlike those, AEs capture both parts in their structure, with the aim that training them becomes easier and faster. In general terms,

[15]: Ranzato et al. (2007), "A Unified Energy-Based Framework for Unsupervised Learning"

AEs are ANNs which produce codifications for input data and are trained so that their decodifications resemble the inputs as closely as possible.

AEs were firstly introduced [16] as a way of conducting pretraining in ANNs. Although mainly developed inside the context of deep learning, not all AE models are necessarily ANNs with multiple hidden layers. As explained below, an AE can be a deep ANN, i.e. in the stacked AEs configuration, or it can be a shallow ANN with a single hidden layer. See Section I.2 for a more detailed introduction to AEs.

[16]: Ballard (1987), "Modular Learning in Neural Networks"

While many machine learning algorithms are able to work with raw input features, it is also true that, for the most part, their behavior is degraded as the number of variables grows. This is mainly due to the problem known as the *curse of dimensionality* [17], as well as the justification for a field of study called feature engineering. Engineering of features started as a manual process, relying in an expert able to decide by observation which variables were better for the task at hand. Notwithstanding, automated feature selection [18] methods were soon available.

[17]: Bellman (1957), *Dynamic Programming*

[18]: Dash et al. (1997), "Feature Selection for Classification"

Feature selection is only one of the approaches to reduce input space dimensionality. Selecting the best subset of input variables is an NP-hard combinatorial problem. Moreover, feature selection techniques usually evaluate each variable independently, but it is known that variables that separately do not provide useful information may do so when they are used together. For this reason other alternatives, primarily feature construction or extraction [19], emerged. In addition to these two denominations, feature selection and feature extraction, when dealing with dimensionality reduction it is also frequent to use other terms. The most common are as follows:

[19]: Liu et al. (1998), *Feature extraction, construction and selection: A data mining perspective*

Feature engineering [20] This is probably the broadest term, encompassing most of the others. Feature engineering can be carried out by manual or automated means, and be based on the selection of original characteristics or the construction of new ones through transformations.

[20]: Domingos (2012), "A few useful things to know about machine learning"

Feature learning [21] It is the denomination used when the process to select among the existing features or construct new ones is automated. Thus, we can perform both feature selection and feature extraction through algorithms such as the ones mentioned below. Despite the use of automatic methods, sometimes an expert is needed to decide which algorithm is the most appropriate depending on

[21]: Bengio et al. (2013), "Representation learning"

data traits, to evaluate the optimum amount of variables to extract, etc.

Representation learning [21] Although this term is sometimes interchangeably used with the previous one, it is mostly used to refer to the use of ANNs to fully automate the feature generation process. Applying ANNs to learn distributed representations of concepts was proposed by Hinton in [22]. Today, learning representations is mainly linked to processing natural language, images and other signals with specific kinds of ANNs, such as CNNs [12].

[21]: Bengio et al. (2013), "Representation learning"

[22]: Hinton (1986), "Learning distributed representations of concepts"

[12]: LeCun et al. (1998), "The Handbook of Brain Theory and Neural Networks"

Feature selection [23] Picking the most informative subset of variables started as a manual process usually in charge of domain experts. It can be considered a special case of feature weighting, as discussed in [24]. Although in certain fields the expert is still an important factor, nowadays the selection of variables is usually carried out using computer algorithms. These can operate in supervised or unsupervised manner. The former approach usually relies on correlation or mutual information between input and output variables [25, 26], while the latter tends to avoid redundancy among features [27]. Feature selection is overall an essential strategy in the data preprocessing [23, 28] phase.

[23]: García et al. (2015), *Data preprocessing in data mining*

[24]: Wettschereck et al. (1997), "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms"

[25]: Hall (1999), "Correlation-based feature selection for machine learning"

[26]: Peng et al. (2005), "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy"

[27]: Mitra et al. (2002), "Unsupervised Feature Selection Using Feature Similarity"

[23]: García et al. (2015), *Data preprocessing in data mining*

[28]: García et al. (2016), "Tutorial on practical tips of the most influential data preprocessing algorithms in data mining"

Feature extraction [29] The goal of this technique is to find a better data representation for the machine learning algorithm intended to use, since the original representation might not be the best one. It can be faced both manually, in which case the *feature construction* term is of common use, and automatically. Some elemental techniques such as normalization, discretization or scaling of variables, as well as basic transformations applied to certain data types*, are also considered within this field. New features can be extracted by finding linear combinations of the original ones, as in PCA (*Principal Component Analysis*) [30, 31] or LDA (*Linear Discriminant Analysis*) [32], as well as nonlinear combinations, like Kernel PCA [33] or Isomap [34]. The latter ones are usually known as *manifold learning* [35] algorithms, and fall in the scope of nonlinear dimensionality reduction techniques [36]. Feature extraction methods can also be categorized as supervised (e.g. LDA) or non-supervised (e.g. PCA).

[29]: Guyon et al. (2006), "An Introduction to Feature Extraction"

[30]: Pearson (1901), "LIII. On lines and planes of closest fit to systems of points in space"

[31]: Hotelling (1933), "Analysis of a complex of statistical variables into principal components"

[32]: Fisher (1938), "The statistical utilization of multiple measurements"

[33]: Schölkopf et al. (1998), "Nonlinear component analysis as a kernel eigenvalue problem"

[34]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

[35]: Cayton (2005), *Algorithms for manifold learning*

[36]: Lee et al. (2007), *Nonlinear dimensionality reduction*

Feature fusion [37] This more recent term has emerged with the

[37]: Mangai et al. (2010), "A survey of decision fusion and feature fusion strategies for pattern classification"

* e.g. Take the original field containing a date and divide it into three new variables, year, month and day.

growth of multimedia data processing by machine learning algorithms, especially images, text and sound. As stated in [37], feature fusion methods aim to combine variables to remove redundant and irrelevant information. Manifold learning algorithms, and especially those based on ANNs, fall into this category.

[37]: Mangai et al. (2010), "A survey of decision fusion and feature fusion strategies for pattern classification"

Among the existing AE models there are several that are useful to perform feature fusion. This is the aim of the most basic one, which can be extended via several regularizations and adaptations to different kinds of data. These options will be explored through the present work, whose aim is to provide the reader with a didactic review on the inner workings of these distinct AE models and the ways they can be used to learn new representations of data.

The following are the main contributions of this paper:

- A proposal of a global taxonomy of AEs dedicated to feature fusion.
- Descriptions of these AE models including the necessary mathematical formulation and explanations.
- A theoretical comparison between AEs and other popular feature fusion techniques.
- A comprehensive review of other AE models as well as their applications.
- A set of guidelines on how to design an AE, and several examples on how an AE may behave when its architecture and parameters are altered.
- A summary of the available software for creating deep learning models and specifically AEs.

Additionally, we provide a case study with the well known dataset MNIST [38], which gives the reader some intuitions on the results provided by an AE with different architectures and parameters. The scrips to reproduce these experiments are provided in a repository, and their use will be further described in Section I.6.

[38]: LeCun et al. (1998), "Gradient-based learning applied to document recognition"

The rest of this paper is structured as follows. The foundations and essential aspects of AEs are introduced in Section I.2, including the proposal of a global taxonomy. Section I.3 is devoted to thoroughly describing the AE models able to operate as feature fusion mechanisms and several models which have further applications. The relationship between these AE models and other feature fusion methods is portrayed in Section I.4, while applications of different kinds of AEs are described in Section I.5. Section I.6 provides a set of guidelines on how to design an AE for the task at hand, followed by the software pieces where it can be implemented, as well as the case study with MNIST data. Concluding remarks can be found in Section I.7. Lastly, an Appendix briefly describes the datasets used through the present work.

I.2 Autoencoder essentials

AEs are ANNs† with a symmetric structure, where the middle layer represents an *encoding* of the input data. AEs are trained to reconstruct their input onto the output layer, while verifying certain restrictions which prevent them from simply copying the data along the network. Although the term *autoencoder* is the most popular nowadays, they were also known as autoassociative neural networks [39], diabolo networks [40] and replicator neural networks [41].

[39]: Kramer (1991), "Nonlinear principal component analysis using autoassociative neural networks"
[40]: Schwenk et al. (1998), "Training methods for adaptive boosting of neural networks"
[41]: Hecht-Nielsen (1995), "Replicator neural networks for universal optimal source coding"

In this section the foundations of AEs are introduced, describing their basic architecture as ANNs as well as the activation functions regularly applied in their layers. Next, AEs are grouped into four types according to their architecture. This is followed by our proposed taxonomy for AEs, which takes into account the properties these induce in codifications. Lastly, a summary of their habitual applications is provided.

General structure

The basic structure of an AE, as shown in Fig. I.1, includes an input x which is mapped onto the encoding y via an encoder, represented as function f. This encoding is in turn mapped to the reconstruction r by means of a decoder, represented as function g.

$$\begin{tikzcd} x \arrow[r, "f"] & y \arrow[r, "g"] & r \end{tikzcd}$$

Figure I.1: General autoencoder structure

This structure is captured in a feedforward neural network. Since the objective is to reproduce the input data on the output layer, both x and r have the same dimension. y, however, can be higher-dimensional or lower-dimensional, depending on the properties desired. The AE can also have as many layers as needed, usually placed symmetrically in the encoder and decoder. Such a neural architecture can be observed in Fig. I.2.

† Strictly speaking not all AEs are ANNs, but here our interest is in those since they are the most common ones.

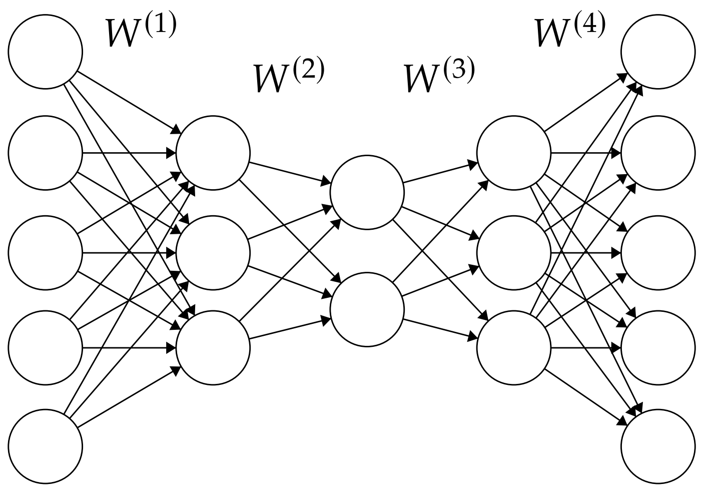
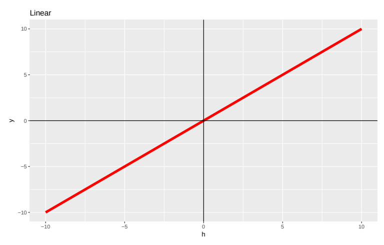


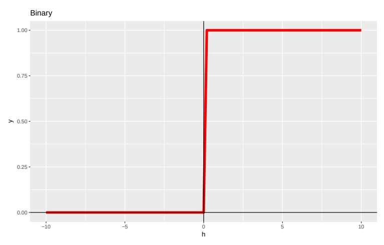
Figure I.2: A possible neural architecture for an autoencoder with a 2-variable encoding layer. W denotes weight matrices.

In this case the encoder is made up of three layers, including the middle encoding one, while the decoder starts in the middle one and also spans three layers.

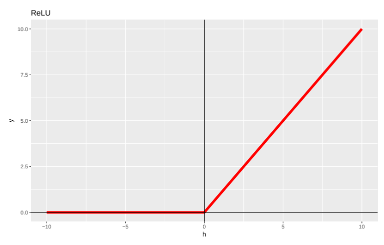
Activation functions of common use in autoencoders



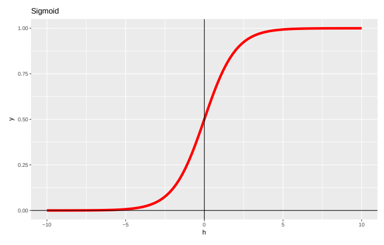
(a) Linear



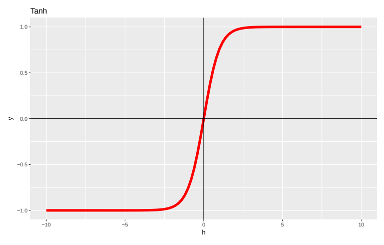
(b) Binary/Boolean



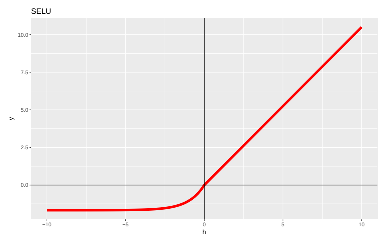
(c) ReLU



(d) Sigmoid



(e) Tanh



(f) SELU

Figure I.3: Common activation functions in ANNs.

A unit located in any of the hidden layers of an ANN receives several inputs from the preceding layer. The unit computes the weighted sum of these inputs and eventually applies a certain operation, the so-called *activation function*, to produce the output. The nonlinearity behavior of most ANNs is founded on the selection of the activation function to be used. Fig. I.3 shows the graphical appearance of six of the most popular ones.

The activation functions shown in the first row are rarely used on AEs when it comes to learning higher level features, since they rarely provide useful representations. An undercomplete AE having one hidden layer made up of k linear activation units (Eq. I.1) and that minimizes the sum of squared errors is known to be equivalent to obtaining the k principal components of the feature space via PCA [42–44]. AEs using binary/boolean activations (Eq. I.2) [45, 46] are mostly adopted for educational uses, as McCulloch and Pitts [2] cells are still used in this context. However, they also have some specific applications, such as data hashing as described in subsection I.5.

[42]: Chicco et al. (2014), "Deep autoencoder neural networks for gene ontology annotation predictions"
[43]: Baldi et al. (1989), "Neural networks and principal component analysis: Learning from examples without local minima"
[44]: Kamyshanska et al. (2013), "On autoencoder scoring"
[45]: Deng et al. (2010), "Binary coding of speech spectrograms using a deep auto-encoder"
[46]: Baldi (2012), "Autoencoders, unsupervised learning, and deep architectures"
[2]: McCulloch et al. (1943), "A logical calculus of the ideas immanent in nervous activity"

$$s_{\text{linear}}(x) = x \tag{I.1}$$
$$s_{\text{binary}}(x) = [x > 0] \tag{I.2}$$

Note that square brackets denote Iverson's convention [47] and evaluate to 0 or 1 according to the truthiness of the proposition.

[47]: Knuth (1992), "Two notes on notation"

Rectified linear units (ReLU, Fig. I.3c, Eq. I.3) are popular in many deep learning models, but it is an activation function that tends to degrade the AE performance. Since it always outputs 0 for negative inputs, it weakens the process of reconstructing the input features onto the outputs. Although they have been successfully used in [48, 49], the authors had to resort to a few detours. A recent alternative which combines the benefits of ReLU while circumventing these obstacles is the SELU function (*Scaled Exponential Linear Units*, Fig. I.3f, Eq. I.4) [50]. There are already some proposals of deep AEs based on SELU such as [51].

[48]: Glorot et al. (2011), "Domain adaptation for large-scale sentiment classification: A deep learning approach"
[49]: Gülçehre et al. (2016), "Knowledge matters: Importance of prior information for optimization"
[50]: Klambauer et al. (2017), "Self-Normalizing Neural Networks"
[51]: Kuchaiev et al. (2018), "Training Deep AutoEncoders for Recommender Systems"

$$s_{\text{relu}}(x) = x[x > 0] \tag{I.3}$$
$$s_{\text{selu}}(x) = \lambda \begin{cases} \alpha e^x - \alpha & x \le 0 \\ x & x > 0 \end{cases}, \text{ where } \lambda > 1 \tag{I.4}$$

Sigmoid functions are undoubtedly the most common activations in AEs. The standard logistic function, popularly known simply as sigmoid (Fig. I.3d, Eq. I.5), is probably the most frequently used. The hyperbolic tangent (Fig. I.3e, Eq. I.6) is also a sigmoid function, but it is symmetric about the origin and presents a steeper slope. According to LeCun [52] the latter should be preferred, since its

[52]: LeCun et al. (1998), "Efficient BackProp"

derivative produces stronger gradients that the former.

$$s_{\text{sigm}}(x) = \sigma(x) = \frac{1}{1+e^x} \tag{I.5}$$

$$s_{\text{tanh}}(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{I.6}$$

When designing AEs with multiple hidden layers, it is possible to use different activation functions in some of them. This would result in AEs combining the characteristics of several of these functions.

Autoencoder groups according to network structure

AEs could be grouped according to disparate principles, such as their structure, the learning algorithm they use, the loss function that guides the update of weights, their activation function or the field they are applied. In this section we focus on the first criterion, while the others will be further covered in following sections.

As explained above, AEs are ANNs with a symmetrical structure. The decoder and the encoder have the same number of layers, with the number of units per layer in reverse order. The encoding layer is shared by both parts. Depending on the dimensionality of the encoding layer, AEs are said to be:

- *Undercomplete*, if the encoding layer has a lower dimensionality than the input. The smaller number of units imposes a restriction, so during training the AE is forced to learn a more compact representation. This is achieved by fusing the original features according to the weights assigned through the learning process.
- *Overcomplete*, otherwise. An encoding layer having the same or more units than the input could allow the AE to simply learn the identity function, copying the input onto the output. To avoid this behavior, usually other restrictions are applied as will be explained later.

Although the more popular AE configuration for dimensionality reduction is undercomplete, an overcomplete AE with the proper restrictions can also produce a compact encoding as explained in subsection I.3.

In addition to the number of units per layer, the structure of an AE is also dependent of the number of layers. According to this factor, an AE can be:

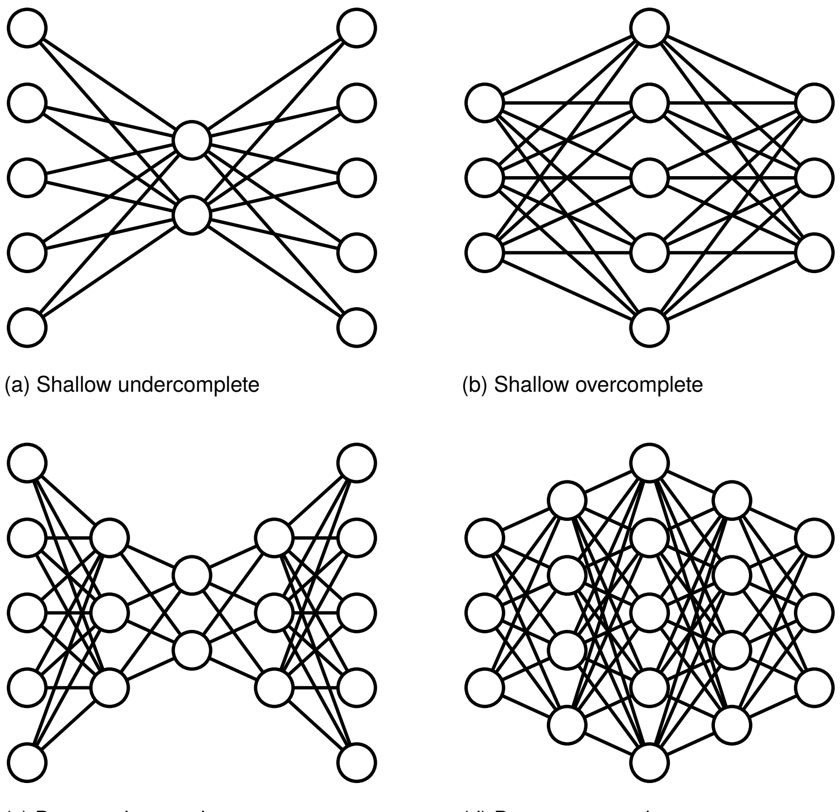


Figure I.4: Autoencoder models according to their structure.

- *Shallow*, when it only comprises three layers (input, encoding and output). It is the simplest AE model, since there is only one hidden layer (the encoding).
- *Deep*, when it has more than one hidden layer. This kind of AE can be trained either layer by layer, as several shallow stacked AEs, or as a deep ANN [53].

[53]: Larochelle et al. (2009), "Exploring strategies for training deep neural networks"

These four types of AEs are visually summarized in Fig. I.4. Shallow AEs are on the top row and deep ones in the bottom, while undercomplete AEs are on the left column and overcomplete on the right one.

Autoencoder taxonomy

As stated before, a taxonomy of AEs can be built according to different criteria. Here the interest is mainly on the properties of the inferred model regarding the feature fusion task. Conforming to this principle, we have elaborated the taxonomy shown in Fig. I.5. As can be seen, there are four main categories in this taxonomy:

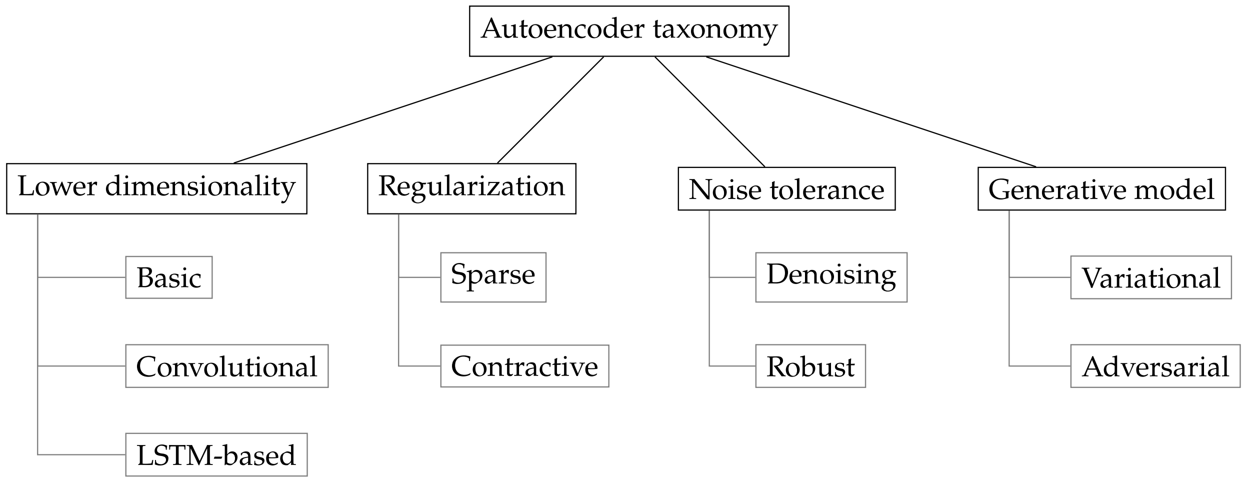


Figure I.5: Taxonomy: most popular autoencoders classified according to the charasteristics they induce in their encodings

Lower dimensionality High-dimensional data can be an issue when using most classifiers and especially shallow neural networks, since they do not perform any kind of high-level feature learning and are then forced to optimize a notable amount of parameters. This task may be eased by just lowering the dimensionality of the data, and this is the aim of the basic AE, which is thoroughly explained in Section I.3. Decreasing the dimensionality of specific types of data, such as images or sequences, can be treated by domain specific AEs detailed in Section I.3.

Regularization Sometimes, learned features are required to present special mathematical properties. AEs can be easily modified in order to reach encodings that verify them. The main regularizations that can be applied to AEs are portrayed in Section I.3.

Noise tolerance In addition to different properties, a desirable trait for the encoded space may be robustness in the face of noisy data. Two distinct approaches to this problem using AEs are gathered in Section I.3.

Generative model The transformation from the original feature space onto the encoded space may not be the main objective of an AE. Occasionally it will be useful to map new samples in the encoded space onto the original features. In this case, a generative model is needed. Those based in AEs are specified in Section I.3.

Usual applications

The term autoencoder is very broad, referring to multiple learning models based on both fully-connected feed-forward ANNs and other types of ANNs, and even models completely unrelated to that structure. Similarly, the application fields of AEs are also varied. In this work we pay attention specifically to AEs whose basic model is that of an ANN. In addition, we are especially interested in those whose objective is the fusion of characteristics by means of nonlinear techniques.

Reducing the dimensionality of a feature space using AEs can be achieved following disparate approaches. Most of them are reviewed in Section I.3, starting with the basic AE model, then advancing to those that include a regularization, that present noise tolerance, etc. The goal is to provide a broad view of the techniques that AEs rely on to perform feature fusion.

Besides feature extraction, which is our main focus, there are AE models designed for other applications such as outlier detection, hashing, data compression or data generation. In Sections I.3 and I.3 some of these AEs will be briefly portrayed, and in Section I.5 many of their applications will be shortly reviewed.

I.3 Autoencoders for feature fusion

As has been already established, AEs are tools originally designed for finding useful representations of data by learning nonlinear ways to combine their features. Usually, this leads to a lower-dimensional space, but different modifications can be applied in order to discover features which satisfy certain requirements. All of these possibilities are discussed in this section, which begins by establishing the foundations of the most basic AE, and later encompasses several diverse variants, following the proposed taxonomy: those that provide regularizations are followed by AEs presenting noise tolerance, generative models are explained afterwards, then some domain specific AEs and finally two variations which do not fit into any previous category.

Basic autoencoder

The main objective of most AEs is to perform a feature fusion process where learned features present some desired traits, such as lower dimensionality, higher sparsity or desirable analytical properties. The resulting model is able to map new instances onto the latent feature space. All AEs thus share a common origin, which may be called the basic AE [54].

[54]: Bourlard et al. (1988), "Auto-association by multilayer perceptrons and singular value decomposition"

The following subsections define the structure of a basic AE, establish their objective function, describe the training process while enumerating the necessary algorithms for this task, and depict how a deep AE can be initialized by stacking several shallow ones.

Structure

The structure of a basic AE, as shown in the previous section, is that of a feed forward ANN where layers are of symmetrical amount of units. Layers need not be symmetrical in the sense of activation functions or weight matrices.

The simplest AE consists of just one hidden layer, and is defined by two weight matrices and two bias vectors:

$$y = f(x) = s_1(W^{(1)}x + b^{(1)}), \tag{I.7}$$
$$r = g(y) = s_2(W^{(2)}y + b^{(2)}), \tag{I.8}$$

where s_1 and s_2 denote the activation functions, which usually are nonlinear.

Deep AEs are the natural extension of this definition to a higher number of layers. We will call the composition of functions in the encoder f, and the composition of functions in the decoder g.

Objective function

AEs generally base their objective function on a per-instance loss function $\mathscr{L} : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$:

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathscr{L}(x, (g \circ f)(x)) \tag{I.9}$$

where f and g are the encoding and decoding functions determined by the weights W and biases b, assuming activation functions are fixed, and S is a set of samples. The objective of an AE is thus to optimize W and b in order to minimize $\mathcal{J}$.

For example, a typical loss function is the mean squared error (MSE):

$$\mathscr{L}_{\text{MSE}}(u, v) = \|u - v\|_2^2 \ . \tag{I.10}$$

Notice that multiplying by constants or performing the square root of the error induced by each instance does not alter the process, since these operations preserve numerical order. As a consequence, the root mean squared error (RMSE) is an equivalent loss metric.

When a probabilistic model is assumed for the input samples, the loss function is chosen as the negative log-likelihood for the example x given the output $(g \circ f)(x)$ [55]. For instance, when input values are binary or modeled as bits, cross-entropy is usually the preferred alternative for the loss function:

[55]: Bengio et al. (2007), "Greedy layer-wise training of deep networks"

$$\mathcal{L}_{\mathrm{CE}}(u, v) = -\sum_{k=1}^{d} u_k \log v_k + (1 - u_k)\log(1 - v_k)\,. \tag{I.11}$$

Training

Usual algorithms for optimizing weights and biases in AEs are stochastic gradient descent (SGD) [56] and some of its variants, such as AdaGrad [57], RMSProp [58] and Adam [59]. Other applicable algorithms which are not based on SGD are L-BFGS and conjugate gradient [60].

[56]: Robbins et al. (1951), "A stochastic approximation method"

[57]: Duchi et al. (2011), "Adaptive subgradient methods for online learning and stochastic optimization"

[58]: Tieleman et al. (2012), "Lecture 6.5-RMSProp"

[59]: Kingma et al. (2015), "Adam: A method for stochastic optimization"

[60]: Ngiam et al. (2011), "On optimization methods for deep learning"

The foundation of these algorithms is the technique of gradient descent [61]. Intuitively, at each step, the gradient of the objective function with respect to the parameters shows the direction of steepest slope, and allows the algorithm to modify the parameters in order to search for a minimum of the function.

[61]: Cauchy (1847), "Méthode générale pour la résolution des systemes d'équations simultanées"

In order to compute the necessary gradients, the backpropagation algorithm [5] is applied. Backpropagation performs this computation by calculating several intermediate terms from the last layer to the first.

[5]: Rumelhart et al. (1986), "Learning representations by back-propagating errors"

AEs, like many other machine learning techniques, are susceptible to overfitting of the training data. To avoid this issue, a regularization term can be added to the objective function which causes a *weight decay* [62]. This improves the generalization ability and encourages smaller weights that produce good reconstructions. Weight decay can be introduced in several ways, but essentially consists in a term depending on weight sizes that will attempt to limit their growth. For example, the resulting objetive function could be

[62]: Krogh et al. (1992), "A simple weight decay can improve generalization"

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda \sum_{i} w_i^2 \tag{I.12}$$

where w_i traverses all the weights in W and λ is a parameter determining the magnitude of the decay.

Further restrictions and regularizations can be applied. A specific constraint that can be imposed is to tie the weight matrices symmetrically, that is, in a shallow AE, to set $W^{(1)} = (W^{(2)})^T$, and the natural extension to deep AEs. This allows to optimize a lower amount of parameters, so the AE can be trained faster, while maintaining the desired architecture.

Stacking

When AEs are deep, the success of the training process relies heavily on a good weight initialization, since there can be from tens to hundreds of thousands of them. This weight initialization can be performed by *stacking* successive shallow AEs [55], that is, training the AE layer by layer in a greedy fashion.

[55]: Bengio et al. (2007), "Greedy layer-wise training of deep networks"

The training process begins by training only the first hidden layer as the encoding of a shallow AE, as shown by the network on the left of Fig. I.6. After this step, the second hidden layer is trained, using the first hidden layer as input layer, as displayed on the right. Inputs are computed via a forward pass of the original inputs through the first layer, with the weights determined during the previous stage. Each successive layer up to the encoding is trained the same way.

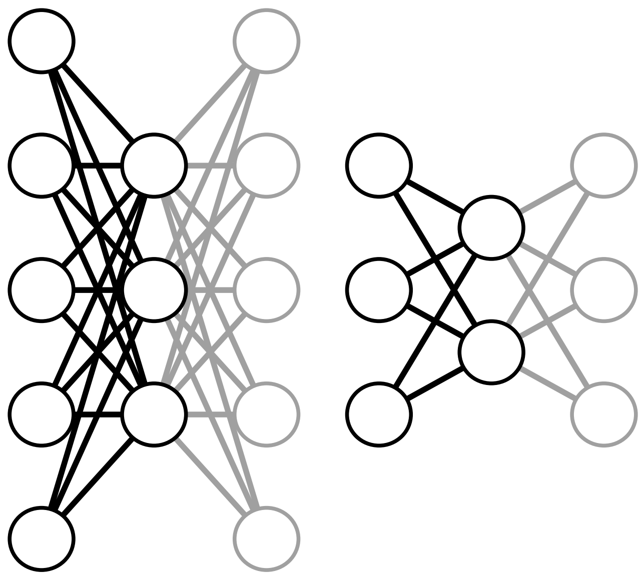


Figure I.6: Greedy layer-wise training of a deep AE with the architecture shown in Fig I.4c. Units drawn in black designate layers of the final AE, and gray ones indicate layers that are not part of the unrolled AE during the fine-tuning phase.

After this layer-wise training, initial weights for all layers preceding and including the encoding layer will have been computed. The AE is now "unrolled", i.e. the rest of layers are added symetrically with weight matrices resulting from transposing the ones from each corresponding layer. For instance, for the AE trained in Fig. I.6, the unrolled AE would have the structure shown in Fig. I.4c.

Finally, a fine-tuning phase can be performed, optimizing the weights by backpropagating gradients through the complete structure with training data.

Regularization

Encodings produced by basic AEs do not generally present any special properties. When learned features are required to verify some desirable traits, some regularizations may be achieved by adding a penalization for certain behaviors Ω to the objetive function:

$$\mathcal{J}(W, b; S) = \sum_{x \in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda \Omega(W, b; S) \,. \tag{I.13}$$

Sparse autoencoder

Sparsity in a representation means most values for a given sample are zero or close to zero. Sparse representations are resembling of the behavior of simple cells in the mammalian primary visual cortex, which is believed to have evolved to discover efficient coding strategies [63]. This motivates the use of transformations of samples into sparse codes in machine learning. A model of sparse coding based on this behavior was first proposed in [64].

[63]: Olshausen et al. (1997), "Sparse coding with an overcomplete basis set: A strategy employed by V1?"

[64]: Olshausen et al. (1996), "Emergence of simple-cell receptive field properties by learning a sparse code for natural images"

Sparse codes can also be overcomplete and meaningful. This was not necessarily the case in basic AEs, where an overcomplete code would be trained to just copy inputs onto outputs.

When sparsity is desired in the encoding generated by an AE, activations of the encoding layer need to have low values in average, which means units in the hidden layer usually do not fire. The activation function used in those units will determine this low value: in the case of sigmoid and ReLU activations, low values will be close to 0; this value will be -1 in the case of tanh, and $-\lambda\alpha$ in the case of a SELU.

[65]: Lee et al. (2008), "Sparse deep belief net model for visual area V2"

The common way to introduce sparsity in an AE is to add a penalty to the loss function, as proposed in [65] for Deep Belief Networks. In order to compare the desired activations for a given unit to the actual ones, these can be modeled as a Bernoulli random variable, assuming a unit can only either fire or not. For a specific input x, let

$$\hat{\rho}_i = \frac{1}{|S|} \sum_{x \in S} f_i(x) \tag{I.14}$$

be the average activation value of an unit in the hidden layer, where $f = (f_1, f_2, \dots f_c)$ and c is the number of units in the encoding. $\hat{\rho}_i$ will be the mean of the associated Bernoulli distribution.

Let ρ be the desired average activation. The Kullback-Leibler divergence [66] between the random variable defined by unit i and the one corresponding to the desired activations will measure how different both distributions are [67]:

[66]: Kullback et al. (1951), "On information and sufficiency"

[67]: Ng (2011), "Sparse autoencoder"

$$\mathrm{KL}(\rho \| \hat{\rho}_i) = \rho \log \frac{\rho}{\hat{\rho}_i} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_i} \ . \tag{I.15}$$

Fig. I.7 shows the penalty caused by Kullback-Leibler divergence for a hidden unit when the desired average activation is $\rho = 0.2$. Notice that the penalty is very low when the average activation is near the desired one, but grows rapidly as it moves away and tends to infinity at 0 and 1.

The resulting penalization term for the objective function is

$$\Omega_{\mathrm{SAE}}(W, b; S) = \sum_{i=1}^{c} \mathrm{KL}(\rho \| \hat{\rho}_i) \ , \tag{I.16}$$

where the average activation value $\hat{\rho}_i$ depends on the parameters of the encoder and the training set S.

There are other modifications that can lead an encoder-decoder architecture to produce a sparse code. For example, applying a sparsifying logistic activation function in the encoding layer of a similar energy-based model, which forces a low activation average [68], or using a Sparse Encoding Symmetric Machine [69] which optimizes a loss function with a different sparsifying penalty.

[68]: Poultney et al. (2007), "Efficient learning of sparse representations with an energy-based model"

[69]: Boureau et al. (2008), "Sparse feature learning for deep belief networks"

Contractive autoencoder

High sensitivity to perturbations in input samples could lead an AE to generate very different encodings. This is usually inconvenient, which is the motivation behind the contractive AE. It achieves local invariance to changes in many directions around the training samples, and is able to more easily discover lower-dimensional manifold structures in the data.

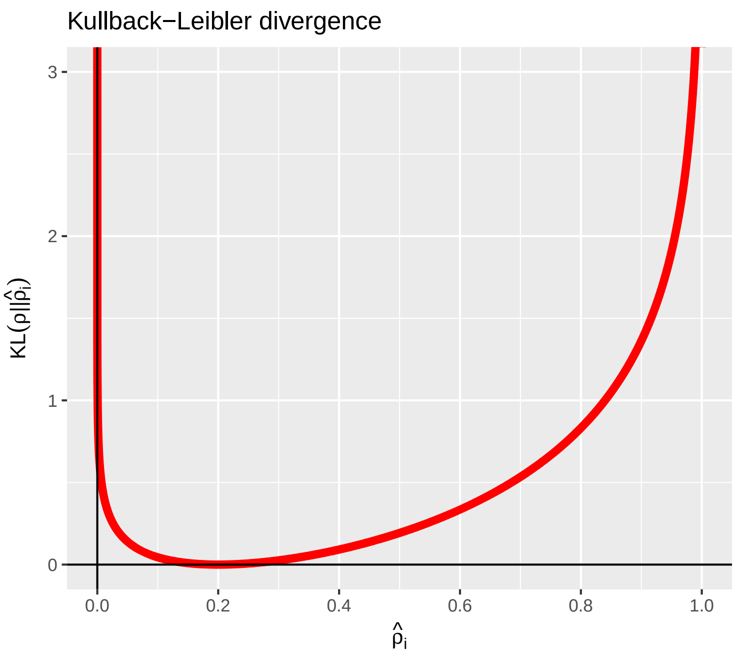


Figure I.7: Values of Kullback-Leibler divergence for a unit with average activation $\hat{\rho}_i$.

Sensitivity for small changes in the input can be measured as the Frobenius norm $\|\cdot\|_F$ of the Jacobian matrix of the encoder J_f:

$$\|J_f(x)\|_F^2 = \sum_{j=1}^{d} \sum_{i=1}^{c} \left(\frac{\partial f_i}{\partial x_j}(x) \right)^2 . \qquad \text{(I.17)}$$

The higher this value is, the more unstable the encodings will be to perturbations on the inputs.

A regularization is built from this measure into the objective function of the contractive AE:

$$\Omega_{\text{CAE}}(W, b; S) = \sum_{x \in S} \|J_f(x)\|_F^2 . \qquad \text{(I.18)}$$

A particular case of this induced contraction is the usage of L2 weight decay with a linear encoder: in this situation, the only way to produce a contraction is to maintain small weights. In the nonlinear case, however, contraction can be encouraged by pushing hidden units to the saturated region of the activation function.

[70]: Rifai et al. (2012), "A Generative Process for sampling Contractive Auto-Encoders"

The contractive AE can be sampled [70], that is, it can generate new instances from the learned model, by using the Jacobian of the encoder to add a small noise to another point and computing its codification. Intuitively, this can be seen as moving small steps along the tangent plane defined by the encoder in a point on the manifold modeled.

Noise tolerance

A standard AE can learn a latent feature space from a set of samples, but it does not guarantee stability in the presence of noisy instances, nor it is able to remove noise when reconstructing new samples. In this section, two variants that tackle this problem are discussed: denoising and robust AEs.

Denoising autoencoder

A denoising AE or DAE [71] learns to generate robust features from inputs by reconstructing partially destroyed samples. The use of AEs for denoising had been introduced earlier [72], but this technique leverages the denoising ability of the AE to build a latent feature space which is more resistant to corrupted inputs, thus its applications are broader than just denoising.

[71]: Vincent et al. (2008), "Extracting and Composing Robust Features with Denoising Autoencoders"

[72]: LeCun (1987), "Modèles connexionnistes de l'apprentissage"

The structure and parameters of a denoising AE are identical to those of a basic AE. The difference here lies in a stochastic corruption of the inputs which is applied during the training phase. The corrupting technique proposed in [71], as illustrated by Fig. I.8, is to randomly choose a fixed amount of features for each training sample and set them to 0. The reconstructions of the AE are however compared to the original, uncorrupted inputs. The AE will be thus be trained to guess the missing values.

[71]: Vincent et al. (2008), "Extracting and Composing Robust Features with Denoising Autoencoders"

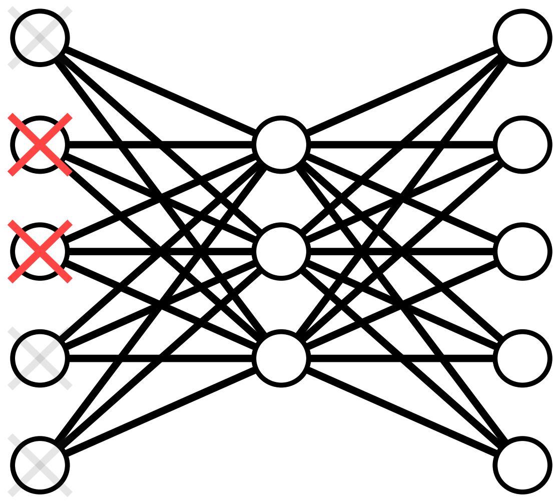


Figure I.8: Illustration of the training phase of a denoising AE. For each input sample, some of its components are randomly selected and set to 0, but the reconstruction error is computed by comparing to the original, non-corrupted data.

Formally, let $q(\tilde{x}|x)$ be a stochastic mapping performing the partial destruction of values described above, the denoising AE recieves

$\tilde{x} \sim q(\tilde{x}|x)$ as input and minimizes

$$\mathcal{J}_{\text{DAE}}(W, b; S) = \sum_{x \in S} \mathbb{E}_{\tilde{x} \sim q(\tilde{x}|x)} \left[\mathcal{L}(x, (g \circ f)(\tilde{x})) \right] . \tag{I.19}$$

A denoising AE does not need further restrictions or regularizations in order to learn a meaningful coding from the data, which means it can be overcomplete if desired. When it has more than one hidden layer, it can be trained layer-wise. For this to be done, uncorrupted inputs are computed as outputs of the previous layers, these are then corrupted and provided to the network. Note that after the denoising AE is trained, it is used to compute higher-level representations without corrupting the input data.

[73]: Vincent et al. (2010), "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion"

The training technique allows for other possible corruption processes, apart from forcing some values to 0 [73]. For instance, additive Gaussian noise

$$\tilde{x} \sim \mathcal{N}(x, \sigma^2 \mathrm{I}) , \tag{I.20}$$

which randomly offsets each component of x with the same variance, or *salt-and-pepper* noise, which sets a fraction of the elements of the input to their minimum or maximum value, according to a uniform distribution.

Robust autoencoder

Training an AE to recover from corrupted data is not the only way to induce noise tolerance in the generated model. An alternative is to modify the loss function used to minimize the reconstruction error in order to dampen the sensitivity to different types of noise.

[74]: Qi et al. (2014), "Robust feature learning by stacked autoencoder with maximum correntropy criterion"

[75]: Liu et al. (2006), "Correntropy"

Robust stacked AEs [74] apply this idea, and manage to be less affected by non-Gaussian noise than standard AEs. They achieve this by using a different loss function based on *correntropy*, a localized similarity measure defined in [75].

$$\mathcal{L}_{\text{MCC}}(u, v) = -\sum_{k=1}^{d} \mathcal{K}_\sigma(u_k - v_k), \tag{I.21}$$

$$\text{where } \mathcal{K}_\sigma(\alpha) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\alpha^2}{2\sigma^2}\right), \tag{I.22}$$

and σ is a parameter for the kernel $\mathcal{K}$.

Correntropy specifically measures the probability density that two events are equal. An advantage of this metric is it being less affected by outliers than MSE. Robust AEs attempt to maximize this measure (equivalently, minimize negative correntropy), which translates in a higher resilience to non-Gaussian noise.

Domain specific autoencoders

The following two AEs are based on the standard type, but are designed to model very specific kinds of data, such as images and sequences.

Convolutional autoencoder [76] Standard AEs do not explicitly consider the 2-dimensional structure when processing image data. Convolutional AEs solve this by making use of convolutional layers instead of fully connected ones. In these, a global weight matrix is used and the convolution operation is applied in order to forward pass values from one layer to the next. The same matrix is flipped over both dimensions and used for the reconstruction phase. Convolutional AEs can also be stacked and used to initialize CNNs [77], which are able to perform classification of images.

[76]: Masci et al. (2011), "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction"

[77]: LeCun et al. (1989), "Backpropagation Applied to Handwritten Zip Code Recognition"

LSTM autoencoder [78] A basic AE is not designed to model sequential data, an LSTM AE achieves this by placing Long-Short-Term Memory (LSTM) [79] units as encoder and decoder of the network. The encoder LSTM reads and compresses a sequence into a fixed-size representation, from which the decoder attempts to extract the original sequence in inverse order. This is especially useful when data is sequential and large, for example video data. A further possible task is to predict the future of the sequence from the representation, which can be achieved by attaching an additional decoder trained for this purpose.

[78]: Srivastava et al. (2015), "Unsupervised learning of video representations using LSTMs"

[79]: Hochreiter et al. (1997), "Long short-term memory"

Generative models

In addition to the models already described, which essentially provide different mechanisms to reduce the dimensionality of a set of variables, the following ones also produce a generative model from the training data. Generative models learn a distribution in order to be able to draw new samples, different from those observed. AEs can generally reconstruct encoded data, but are not necessarily able to build meaningful outputs from arbitrary encodings. Variational and adversarial AEs learn a model of the data from which new instances can be generated.

Variational autoencoder [80] This kind of AE applies a variational Bayesian [81] approach to encoding. It assumes that a latent, unobserved random variable **y** exists, which by some random process leads to the observations, **x**. Its objective is thus to approximate the distribution of the latent variable given the observations. Variational AEs replace deterministic functions in the encoder and decoder by stochastic mappings, and compute the objective function in virtue of the density functions of the random variables:

[80]: Kingma et al. (2013), "Auto-encoding variational bayes"

[81]: Fox et al. (2012), "A tutorial on variational Bayesian inference"

$$\mathscr{L}_{\mathrm{VAE}}(\theta,\phi;\mathbf{x}) = \mathrm{KL}(q_\phi(\mathbf{y}|\mathbf{x})\|p_\theta(\mathbf{y})) - \mathbb{E}_{q_\phi(\mathbf{y}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{y})\right] \ , \quad \text{(I.23)}$$

where q is the distribution approximating the true latent distribution of $\mathbf{y}$, and θ, ϕ are the parameters of each distribution. Since variational AEs allow sampling from the learned distribution, applications usually involve generating new instances [82, 83].

[82]: Dosovitskiy et al. (2016), "Generating images with perceptual similarity metrics based on deep networks"

[83]: Rezende et al. (2014), "Stochastic Backpropagation and Approximate Inference in Deep Generative Models"

[84]: Makhzani et al. (2015), "Adversarial autoencoders"

[85]: Goodfellow et al. (2014), "Generative adversarial nets"

Adversarial autoencoder [84] It brings the concept of Generative Adversarial Networks [85] to the field of AEs. It models the encoding by imposing a prior distribution, then training a standard AE and, concurrently, a discriminative network trying to distinguish codifications from samples from the imposed prior. Since the generator (the encoder) is trained to fool the discriminator as well, encodings will tend to follow the imposed distribution. Therefore, adversarial AEs are also able generate new meaningful samples.

Other generative models based on similar principles are Variational Recurrent AEs [86], PixelGAN AEs [87] and Adversarial Symmetric Variational AEs [88].

[86]: Fabius et al. (2015), "Variational recurrent auto-encoders"

[87]: Makhzani et al. (2017), "PixelGAN Autoencoders"

[88]: Pu et al. (2017), "Adversarial symmetric variational autoencoder"

Other autoencoders farther from feature fusion

As can be seen, AEs can be easily altered to achieve different properties in their encoding. The following are some AEs which do not fall into any previous category.

Relational autoencoder Basic AEs do not explicitly consider the possible relations among instances. The relational AE [89] modifies the objective function to take into account the fidelity of the reconstruction of relationships among samples. Instead of just adding a penalty term, the authors propose a weighted sum of the sample reconstruction error and the relation reconstruction error. Notice

[89]: Meng et al. (2017), "Relational autoencoder for feature extraction"

that this is not the only variation named "relational autoencoder" by its authors, different but identically named models are commented in sections I.3 and I.5.

Discriminative autoencoder Introduced in [90], the discriminative AE uses the class information of instances in order to build a manifold where positive samples are gathered and negative samples are pushed away. As a consequence, this AE performs better reconstruction of positive instances than negative ones. It achieves this by optimizing a different loss function, specifically the hinge loss function used in metric learning. The main objective of this model is object detection.

[90]: Razakarivony et al. (2014), "Discriminative autoencoders for small targets detection"

Autoencoder-based architectures for feature learning

The basic AE can also be used as building block or inspiration for other, more complex architectures dedicated to feature fusion. This section enumerates and briefly introduces the most relevant ones.

Autoencoder trees [91] (Fig. I.9a) are encoder-decoder architectures, inspired by neural AEs, where the encoder as well as the decoder are actually decision trees. These trees use soft decision nodes, which means they propagate instances to all their children with different probabilities.

[91]: İrsoy et al. (2017), "Unsupervised feature extraction with autoencoder trees"

A dual-autoencoder architecture [92] (Fig. I.9b) attempts to learn two latent representations for problems where variables can be treated as instances and viceversa, e.g. predicting customers' recommendations of items. These two representations are linked by an additional term in the objective function which minimizes their deviation from the training data.

[92]: Zhuang et al. (2017), "Representation learning via Dual-Autoencoder for recommendation"

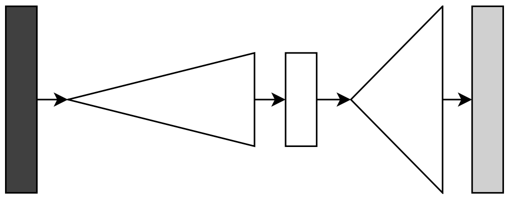
The relational or "cross-correlation" AE defined in [93] incorporates layers where units are combined by multiplication instead of by a weighted sum. This allows it to represent co-ocurrences among components of its inputs.

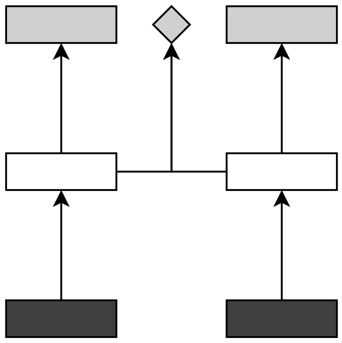
[93]: Memisevic (2011), "Gradient-based learning of higher-order image features"

A recursive AE [94] (Fig. I.9c) is a tree-like architecture built from AEs, in which new pieces of input are introduced as the model gets deeper. A standard recursive AE attempts to reconstruct only the direct inputs of each encoding layer, whereas an unfolding recursive AE [95] reconstructs all previous inputs from each encoding layer. This architecture is designed to model sentiment in sentences.

[94]: Socher et al. (2011), "Semi-supervised recursive autoencoders for predicting sentiment distributions"

[95]: Socher et al. (2011), "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection"


(a) AE tree. Triangles represent decision trees.


(b) Dual AE. The encodings are coupled by an additional penalty term, represented as a diamond.

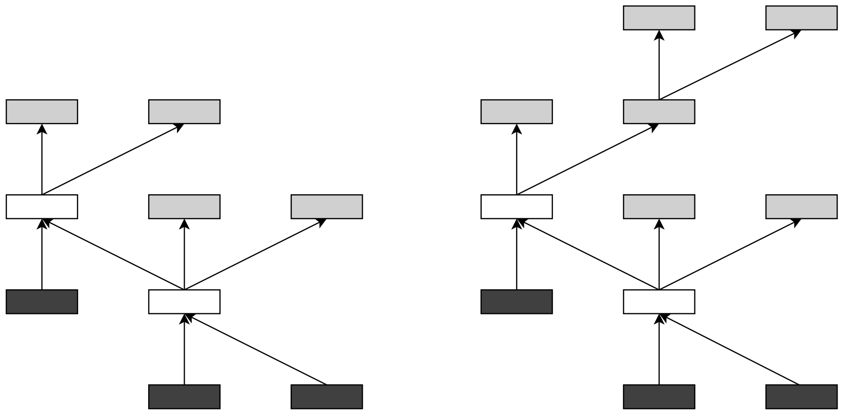

(c) Recursive AE (left), unfolded recursive AE (right)

Figure I.9: Illustrations of autoencoder-based architectures. Each rectangle represents a layer, dark gray fill represents an input, light gray represents output layers and white objects represent hidden layers.

I.4 Comparison to other feature fusion techniques

AEs are only several of a very diverse range of feature fusion methods [37]. These can be grouped according to whether they perform supervised or unsupervised learning. In the first case, they are usually known as *distance metric learning* techniques [96]. Some adversarial AEs, as well as AEs preserving class neighborhood structure [97], can be sorted into this category, since they are able to make use of the class information. However, this section focuses on the latter case, since most AEs are unsupervised and therefore share more similarities with this kind of methods.

[37]: Mangai et al. (2010), "A survey of decision fusion and feature fusion strategies for pattern classification"

[96]: Wang et al. (2015), "Survey on distance metric learning and dimensionality reduction in data mining"

[97]: Salakhutdinov et al. (2007), "Learning a nonlinear embedding by preserving class neighbourhood structure"

A dimensionality reduction technique is said to be *convex* if it optimizes a function which does not have any local optima, and it is *nonconvex* otherwise [98]. Therefore, a different classification of these techniques is into convex and nonconvex approaches. AEs fall into the nonconvex group, since they can attempt to optimize disparate objective functions, and these may present more than one optimum. AEs are also not restrained to the dimensionality reduction domain, since they can produce sparse codes and other meaningful overcomplete representations.

[98]: Van Der Maaten et al. (2009), *Dimensionality reduction: a comparative review*

Lastly, feature fusion procedures can be carried out by means of linear or nonlinear transformations. In this section, we aim to summarize the main traits of the most relevant approaches in both of these situations, and compare them to AEs.

Linear approaches

Principal component analysis is a statistical technique developed geometrically by Pearson [30] and algebraically by Hotelling [31]. It consists in the extraction of the *principal components* of a vector of random variables. Principal components are linear combinations of the original variables in a specific order, so that the first one has maximum variance, the second one has maximum possible variance while being uncorrelated to the first (equivalently, orthogonal), the third has maximum possible variance while being uncorrelated to the first and second, and so on. A modern analytical derivation of principal components can be found in [99].

[30]: Pearson (1901), "LIII. On lines and planes of closest fit to systems of points in space"

[31]: Hotelling (1933), "Analysis of a complex of statistical variables into principal components"

[99]: Jolliffe (1986), *Principal component analysis*

The use of PCA for dimensionality reduction is very common, and can lead to reasonably good results. It is known that AEs with linear activations that minimize the mean quadratic error learn the principal components of the data [43]. From this perspective, AEs can be regarded as generalizations of PCA. However, as opposed to PCA, AEs can learn nonlinear combinations of the variables and even overcomplete representations of data.

[43]: Baldi et al. (1989), "Neural networks and principal component analysis: Learning from examples without local minima"

Fig. I.10 shows a particular occurrence of these facts in the case of the MNIST dataset [38]. Row 1 shows several test samples and the rest display reconstructions built by PCA and some AEs. As can be inferred from rows 2 and 3, linear AEs which optimize MSE learn an approximation of PCA. However, just by adjusting the activation functions and the objective function of the neural network one can obtain superior results (row 4). Improvements over the standard AE such as the robust AE (row 5) also provide higher quality in their reconstructions.

[38]: LeCun et al. (1998), "Gradient-based learning applied to document recognition"

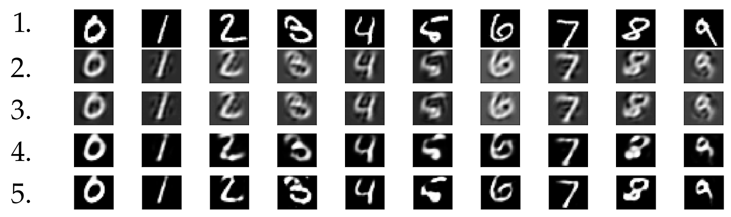


Figure I.10: Row 1 shows test samples, second row corresponds to PCA reconstructions, the third one shows those from a linear AE optimizing MSE, row 4 displays reconstructions from a basic AE with tanh activation and cross-entropy as loss function, and last row corresponds to a robust AE.

A procedure similar to PCA but from a different theoretical perspective is Factor Analysis (FA) [100], which assumes a set of latent variables or *factors* which are not observable but are linearly combined to produce the observed variables. The difference between PCA and FA is similar to that between the basic AE and the variational AE: the latter assumes that hypothetical, underlying variables exist and cause the observed data. Variational AEs and FA attempt to find the model that best describes these variables, whereas the basic AE and PCA only aim for a lower-dimensional representation.

[100]: Jolliffe (1986), "Principal Component Analysis and Factor Analysis"

Linear Discriminant Analysis (LDA) [101] is a supervised statistical method to find linear combinations of features that achieve good separation of classes. It makes some assumptions of normality and homoscedasticity over the data, and projects samples onto new coordinates that best discriminate classes. It can be easily seen that AEs are very different in theory to this method: they usually perform unsupervised learning, and they do not necessarily make previous assumptions of the data. In contrast, AEs may not find the best separation of classes but they might encode further meaningful information from the data. Therefore, these techniques may be convenient, each in very different types of problems.

[101]: Fisher (1936), "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS"

Nonlinear approaches

Kernel PCA [33] is an extension of PCA which applies kernel methods in order to extract nonlinear combinations of variables. Since principal components can be computed by projecting samples onto the eigenvectors of the covariance matrix, the kernel trick can be applied in order to calculate the covariance matrix of the data in a higher-dimensional space, given by the kernel function. Therefore, kernel PCA can compute nonlinear combinations of variables and overcomplete representations. The choice of kernel, however, can determine the success of the method and may behave differently with each problem, and hence AEs are a more general and easily applicable framework for nonlinear feature fusion.

[33]: Schölkopf et al. (1998), "Nonlinear component analysis as a kernel eigenvalue problem"

Multidimensional Scaling (MDS) [102] is a well known technique

[102]: Torgerson (1952), "Multidimensional scaling: I. Theory and method"

and a foundation for other algorithms. It consists in finding new coordinates in a lower-dimensional space, while maintaining relative distances among data points as accurately as possible. For this to be achieved, it computes pairwise distances among points and then estimates an origin or zero point for these, which allows to transform relative distances into absolute distances that can be fitted into a real Euclidean space. Sammon mapping [103] modifies the classical cost function of MDS, in an attempt to similarly weigh retaining large distances as well as small ones. It achieves better preservation of local structure than classic MDS, at the cost of giving more importance to very small distances than large ones.

[103]: Sammon (1969), "A nonlinear mapping for data structure analysis"

The approach of MDS to nonlinear feature fusion is opposite to that of AEs, which generally do not directly take into account distances among pairs of samples, and instead optimize a global measure of fitness. However, the objective function of an AE can be combined with that of MDS in order to produce a nonlinear embedding which considers pairwise distances among points [104].

[104]: Yu et al. (2013), "Embedding with autoencoder regularization"

Isomap [34] is a manifold learning method which extends MDS in order to find coordinates that describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances between the rest of points. In addition, Locally Linear Embedding (LLE) [105] has a similar goal, to learn a manifold which preserves neighbors, but a very different approach: it linearly reconstructs each point from its neighbors in order to maintain the local structure of the manifold.

[34]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

[105]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

Both of these techniques can be compared to the contractive AE, as it also attempts to preserve the local behavior of the data in its encoding. Denoising AEs may also be indirectly forced to learn manifolds, when they exist, and corrupted examples will be projected back onto their surface [73]. However, AEs are able to map new instances onto the latent space after they have been trained, a task Isomap and LLE are not designed for.

[73]: Vincent et al. (2010), "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion"

[106]: Belkin et al. (2003), "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation"

Laplacian Eigenmaps [106] is a framework aiming to retain local properties as well. It consists in constructing an adjacency graph where instances are nodes and neighbors are connected by edges. Then, a weight matrix similar to an adjacency matrix is built. Last, eigenvalues and eigenvectors are obtained for the Laplacian matrix associated to the weight matrix, and those eigenvectors (except 0) are used to compute new coordinates for each point. As previously mentioned, AEs do not usually consider the local structure of the data, except for contractive AEs and further regularizations which incorporate measures of local properties into the objective function, such as Laplacian AEs [107].

[107]: Jia et al. (2015), "Laplacian Auto-Encoders: An explicit learning of nonlinear data manifold"

[108]: Goodfellow et al. (2016), "Deep Learning"

[109]: Smolensky (1986), *Information processing in dynamical systems: Foundations of harmony theory*

A Restricted Boltzmann Machine (RBM) [108], introduced originally as *harmonium* in [109], is an undirected graphical model, with one visible layer and one hidden layer. They are defined by a joint probability distribution determined by an energy function. However, computing probabilities is unfeasible since the distribution is intractable, and they have been proved to be hard to simulate [110]. Instead, Contrastive Divergence [111] is used to train an RBM. RBMs are an alternative to AEs for greedy layer-wise initialization of weights in ANNs including AEs. AEs, however, are trained with more classical methods and are more easily adaptable to different tasks than RBMs.

[110]: Long et al. (2010), "Restricted Boltzmann machines are hard to approximately evaluate or simulate"

[111]: Hinton (2002), "Training Products of Experts by Minimizing Contrastive Divergence"

I.5 Applications in feature learning and beyond

The ability of AEs to perform feature fusion is useful for easing the learning of predictive models, improving classification and regression results, and also for facilitating unsupervised tasks that are harder to conduct in high-dimensional spaces, such as clustering. Some specific cases of these applications are portrayed within the following subsections, including:

- Classification: reducing or transforming the training data in order to achieve better performance in a classifier.
- Data compression: training AEs for specific types of data to learn efficient compressions.
- Detection of abnormal patterns: identification of discordant instances by analyzing generated encodings.
- Hashing: summarizing input data onto a binary vector for faster search.
- Visualization: projecting data onto 2 or 3 dimensions with an AE for graphical representation.
- Other purposes: further applications of AEs.

Classification

Using any of the AE models described in Section I.3 to improve the output of a classifier is something very common nowadays. Here only a few but very representative case studies are referenced.

Classifying tissue images to detect cancer nuclei is a very complicated accomplishment, due to the large size of high-resolution pathological images and the high variance of the fundamental traits of these nuclei, e.g. its shape, size, etc. The authors of [112] introduce a method, based on stacked DAEs to produce higher level and more compact features, which eases this task.

[112]: Xu et al. (2016), "Stacked Sparse Autoencoder (SSAE) for Nuclei Detection on Breast Cancer Histopathology Images"

Multimodal/Multiview learning [113] is a rising technique which also found considerable support in AEs. The authors of [114] present a general procedure named *Orthogonal Autoencoder for Multi-View*. It is founded on DAEs to extract private and shared latent feature spaces, with an added orthogonality constraint to remove unnecessary connections. In [115] the authors propose the MSCAE (*Multimodal Stacked Contractive Autoencoder*), an application-specific model fed with text, audio and image data to perform multimodal video classification.

[113]: Xu et al. (2013), "A survey on multi-view learning"

[114]: Ye et al. (2016), "Learning Multiple Views with Orthogonal Denoising Autoencoders"

[115]: Liu et al. (2016), "Multimodal video classification with stacked contractive autoencoders"

Multilabel classification [116] (MLC) is another growing machine learning field. MLC algorithms have to predict several outputs (labels) linked to each input pattern. These are usually defined by a high-dimensional feature vector and a set of labels as output which tend to be quite large as well. In [117] the authors propose an AE-based method named C2AE (*Canonical Correlated AutoEncoder*), aimed to learn a compressed feature space while establishing relationships between the input and output spaces.

[116]: Herrera et al. (2016), *Multilabel Classification. Problem analysis, metrics and techniques*

[117]: Yeh et al. (2017), "Learning Deep Latent Space for Multi-Label Classification"

Text classification following a semi-supervised approach by means of AEs is introduced in [118]. A model called SSVAE (*Semi-supervised Sequential Variational Autoencoder*) is presented, mixing a Seq2Seq [119] structure and a sequential classifier. The authors state that their method outperforms fully supervised methods.

[118]: Xu et al. (2017), "Variational Autoencoder for Semi-Supervised Text Classification"

[119]: Kalchbrenner et al. (2013), "Recurrent Continuous Translation Models."

Classifiers based on AEs can be grouped in ensembles in order to gain expressive power, but some diversity needs to be introduced. Several means of doing so, as well as a proposal for Stacked Denoising Autoencoding (SDAE) classifiers can be found in [120]. This method has set a new performance record in MNIST classification.

[120]: Alvear-Sandoval et al. (2018), "On building ensembles of stacked denoising auto-encoding classifiers and their further improvement"

Data compression

Since AEs are able to reconstruct the inputs given to them, an obvious application would be compressing large amounts of data. However, as we already know, the AE output is not perfect, but an approximate reconstruction of the input. Therefore, it is useful only when lossy compression is permissible.

This is the usual scenario while working with images, hence the popularity of the JPEG [121] graphic file format. It is therefore not surprising that AEs have been successfully applied to this task. This

[121]: Wallace (1992), "The JPEG still picture compression standard"

is the case of [122], where the performance of several AE models compressing mammogram image patches is analyzed. A less specific goal can be found in [123]. It proposes a model of AE named SWTA AE (*Stochastic Winner-Take-All Auto-Encoder*), a variation of the sparse AE model, aimed to work as a general method able to achieve a variable ratio of image compression.

[122]: Tan et al. (2011), "Using Autoencoders for Mammogram Compression"

[123]: Dumas et al. (2017), "Image Compression with Stochastic Winner-Take-All Auto-Encoder"

Although images could be the most popular data compressed by means of AEs, these have also demonstrated their capacity to work with other types of information as well. For instance:

- In [124] the authors suggest the use of AEs to compress biometric data, such as blood pressure or heart rate, retrieved by wearable devices. This way battery life can be extended while time transmission of data is reduced.
- Language compression is the goal of ASC (*Autoencoding Sentence Compression*), a model introduced in [125]. It is founded on a variational AE, used to draw sentences from a language modeled with a certain distribution.
- High-resolution time series of data, such as measurements taken from service grids (electricity, water, gas, etc.), tend to need a lot of space. In [126] the APRA (*Adaptive Pairwise Recurrent Encoder*) model is presented, combining an AE and a LSTM to successfully compress this kind of information.

[124]: Del Testa et al. (2015), "Lightweight lossy compression of biometric patterns via denoising autoencoders"

[125]: Miao et al. (2016), "Language as a latent variable: Discrete generative models for sentence compression"

[126]: Hsu (2017), "Time Series Compression Based on Adaptive Piecewise Recurrent Autoencoder"

Lossy compression is assumed to be tolerable in all these scenarios, so the approximate reconstruction process of the AE does not hinder the main objective in each case.

Detection of abnormal patterns

Abnormal patterns are samples present in the dataset that clearly differ from the remaining ones. The distinction between *anomalies* and *outliers* is usually found in the literature, although according to Aggarwal [127] these terms, along with *deviants*, *discordants* or *abnormalities*, refer to the same concept.

[127]: Aggarwal (2013), "An introduction to outlier analysis"

The telemetry obtained from spacecrafts is quite complex, made up of hundreds of variables. The authors of [128] propose the use of basic and denoising AEs for facing anomaly detection taking advantage of the nonlinear dimensionality reduction ability of these models. The comparison with both PCA and Kernel PCA demonstrates the superiority of AEs in this task.

[128]: Sakurada et al. (2014), "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction"

The technique introduced in [129] aims to improve the detection of outliers. To do so, the authors propose to create ensembles of

[129]: Chen et al. (2017), "Outlier Detection with Autoencoder Ensembles"

AEs with random connections instead of fully connected layers. Their model, named RandNet (*Randomized Neural Network for Outlier Detection*), is compared against four classic outlier detection methods achieving an outstanding performance.

A practical application of abnormal pattern detection with AEs is the one proposed in [130]. The authors of this work used a DAE, trained with a benchmark dataset, to identify fake twitter accounts. This way legitimate followers can be separated of those that are not.

[130]: Castellini et al. (2017), "Fake Twitter followers detection by denoising autoencoder"

Hashing

Hashing [131] is a very common technique in computing, mainly to create data structures able to offer constant access time to any element (*hash tables*) and to provide certain guarantees in cryptography (*hash values*). A special family of hash functions are those known as *Locality Sensitive Hashing* (LSH) [132]. They have the ability to map data patterns to lower dimensional spaces while maintaining some topological traits, such as the relative distance between these patterns. This technique is very useful for some applications, such as similar document retrieval. AEs can be also applied in these same fields.

[131]: Chi et al. (2017), "Hashing Techniques: A Survey and Taxonomy"

[132]: Gionis et al. (1999), "Similarity Search in High Dimensions via Hashing"

Salakhutdinov and Hinton demonstrated in [133] how to perform what they call *semantic hashing* through a multi-layer AE. The fundamental idea is to restrict the values of the encoding layer units so that they are binary. In the example proposed in this study that layer has 128 or 20 units, sequences of ones and zeroes that are interpreted as an address. The aim is to facilitate the retrieval of documents, as noted above. The authors show how this technique offers better performance than the classic TF-IDF [134] or LSH.

[133]: Salakhutdinov et al. (2009), "Semantic hashing"

[134]: Salton et al. (1983), "Extended Boolean information retrieval"

Although the approach to generate the binary AE is different from the previous one, since they achieve hashing with binary AEs helped by MAC (*Method of Auxiliary Coordinates*) [135], the proposal in [136] is quite similar. The encoding layer produces a string of zeroes and ones, used in this case to conduct fast search of similar images in databases.

[135]: Carreira-Perpinan et al. (2014), "Distributed optimization of deeply nested systems"

[136]: Carreira-Perpinán et al. (2015), "Hashing with binary autoencoders"

Data visualization

Understanding the nature of a given dataset can be a complex task when it posesses many dimensions. Data visualization techniques [137] can help analyze the structure of the data. One way of visualiz-

[137]: Fayyad et al. (2002), *Information visualization in data mining and knowledge discovery*

ing all instances in a dataset is to project it onto a lower-dimensional space which can be represented graphically.

A particular useful case of AEs are those with a 2 or 3-variable encoding [138]. This allows the generated codifications of samples to be displayed in a graphical representation such as the one in Fig. I.11.

[138]: Hinton (2006), "Reducing the Dimensionality of Data with Neural Networks"

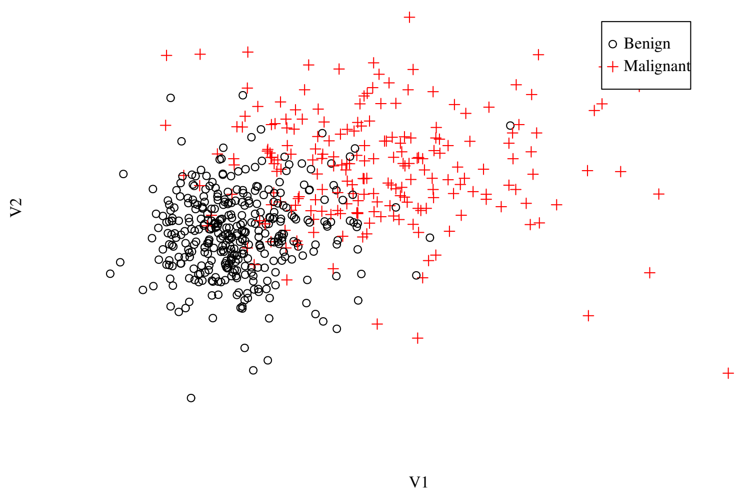


Figure I.11: Example visualization of the codifications of a Cancer dataset generated with a basic AE with weight decay.

The original data [139] has 30 variables describing each pattern. Each data point is linked to one of two potential cancer diagnosis (classes), Benign and Malignant. These have been used in Fig. I.11 to better show the separation between the two classes, but the V1 and V2 variables have been produced by the AE in an unsupervised fashion. Different projections could be obtained by adjusting the AE parameters.

[139]: Mangasarian et al. (1995), "Breast cancer diagnosis and prognosis via linear programming"

Other applications of autoencoders

Beyond the specific applications within the four previous categories, which can be considered as usual in terms of the use of AEs, these find to be useful in many other cases. The following are just a few specific examples.

Holographic images [140] are a useful resource to store information in a fast way. However, retrieval of data has to face a common obstacle as is image degradation by the presence of speckle noise. In [141] an AE is trained with original holographic images as well as with degraded images, aiming to have a decoder able to reconstruct deteriorated examples. The denoising of images is also the goal of the

[140]: Poon (2006), *Digital holography and three-dimensional display: Principles and Applications*

[141]: Shimobaba et al. (2017), "Autoencoder-based holographic image restoration"

method introduced in [142], although in this case they are medical images and the AE method is founded on convolutional denosing AEs.

[142]: Gondara (2016), "Medical Image Denoising Using Convolutional Denoising Autoencoders"

The use of AEs to improve automatic speech recognition (ASR) systems has been also studied in late years. The authors of [143] rely on a DAE to reduce the noise and thus perform speech recognition enhancement. Essentially, the method gives the deep DAE noisy speech samples as inputs while the reference outputs are clean. A similar procedure is followed in [144], although in this case the problem present in the speech samples is reverberation. ASR is specially challenging when faced with whispered speech, as described in [145]. Once more, a deep DAE is the tool to improve results from classical approaches.

[143]: Lu et al. (2013), "Speech enhancement based on deep denoising autoencoder"

[144]: Ishii et al. (2013), "Reverberant speech recognition based on denoising autoencoder"

[145]: Grozdić et al. (2017), "Whispered Speech Recognition Using Deep Denoising Autoencoder and Inverse Filtering"

The procedure to curate biological databases is very expensive, so usually machine learning methods such as SVD (*Singular Value Decomposition*) [146] are applied to help in the process. In [42] this classical approach is compared with the use of deep AEs, reaching as conclusion that the latter is able to improve the results.

[146]: Golub et al. (1970), "Singular value decomposition and least squares solutions"

[42]: Chicco et al. (2014), "Deep autoencoder neural networks for gene ontology annotation predictions"

The authors of [147] aim to perform multimodal fusion by means of deep AEs, specifically proposing a Multimodal Deep Autoencoder (MDA). The goal is to perform human pose recovery from video [148]. To do so, two separate AEs are used to obtain high-level representations of 2D images and 3D human poses. Connecting these two AEs, a two-layer ANN carries out the mapping between the two representations.

[147]: Hong et al. (2015), "Multimodal Deep Autoencoder for Human Pose Recovery"

[148]: Zhou et al. (2014), "Vision-based pose estimation from points with unknown correspondences"

Tagging digital resources, such as movies and products [149] or even questions in forums [150], helps the users in finding the information they are interested in, hence the importance in designing tag recommendation systems. The foundation of the approach in [151] is an AE variation named RSDAE (*Relational Stacked Denoising Autoencoder*). This AE works as a graphical model, combining the learning of high-level features with relationships among items.

[149]: Vig et al. (2009), "Tagsplanations: explaining recommendations using tags"

[150]: Charte et al. (2015), "QUINTA: a question tagging assistant to improve the answering ratio in electronic forums"

[151]: Wang et al. (2015), "Relational Stacked Denoising Autoencoder for Tag Recommendation"

AEs are also scalable to diverse applications with big data, where the stacking of networks acquires notable importance [152]. Multi-modal AEs and Tensor AEs are some examples of variants developed in this field.

[152]: Zhang et al. (2018), "A survey on deep learning for big data"

I.6 Guidelines, software and examples on autoencoder design

This section attempts to guide the user along the process of designing an AE for a given problem, reviewing the range of choices the user has and their utility, then summarizing the available software for deep learning and outlining the steps needed to implement an AE. It also provides a case study with the MNIST dataset where the impact of several parameters of AEs is explored, as well as different AE types with identical parameter settings.

Guidelines

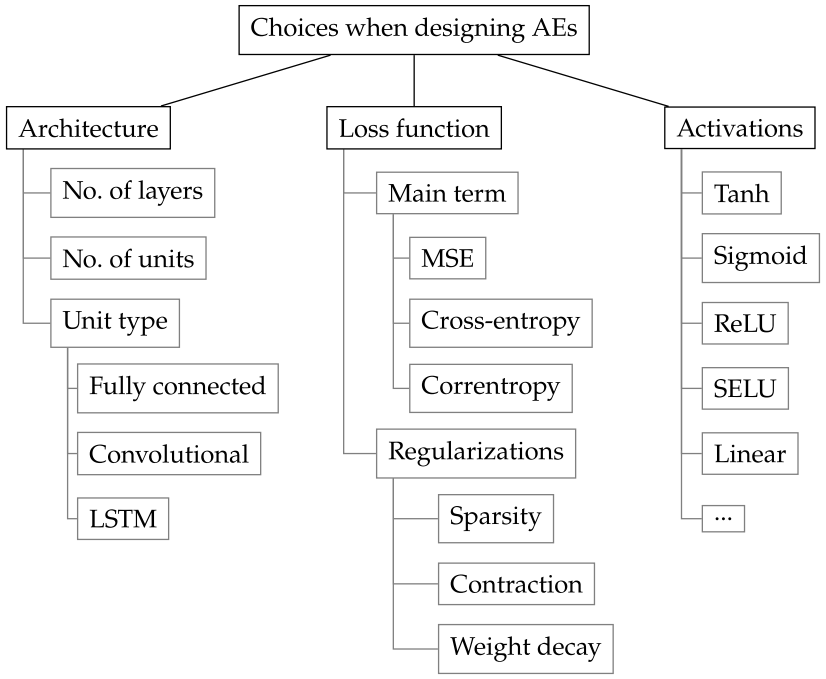


Figure I.12: Summary of choices when designing an AE

When building an AE for a specific task, it is convenient to take into consideration the modifications studied in Section I.3. There is no need to choose just one of those, most of them can actually be combined in the same AE. For instance, one could have a stacked denoising AE with weight decay and sparsity regularizations. A schematic summary of these can be viewed in Fig. I.12.

Architecture Firstly, one must define the structure of the AE, especially the length of the encoding layer. This is a fundamental step that will determine whether the training process can lead it to a good codification. If the length of the encoding is proportionally very low with respect to the number of original variables, training a deep stacked AE should be considered. In addition, convolutional layers are generally better performant with image data, whereas LSTM encoders and decoders would be preferable when modeling sequences. Otherwise, fully connected layers should be chosen.

Activations and loss function Activation functions that will be applied within each layer have to be decided according to the loss function which will be optimized. For example, a sigmoid-like function such as the logistic or tanh is generally a reasonable choice for the encoding layer, the latter being usually preferred due to its greater gradients. This does not need to coincide with the activation in the output layer. Placing a linear activation or ReLU at the output can be sensible when using mean squared error as reconstruction error, while a logistic activation would be better combined with the cross-entropy error and normalized data, since it outputs values between 0 and 1.

Regularizations On top of that, diverse regularizations may be applied that will lead the AE to improve its encoding following certain criteria. It is generally advisable to add a small weight decay in order to prevent it from overfitting the training data. A sparse codification is useful in many cases and adds more flexibility to the choice of structure. Additionally, a contraction regularization may be valuable if the data forms a lower-dimensional manifold.

As seen in previous sections, AEs provide high flexibility and can be further modified for very different applications. In the case that the standard components do not fit the desired behavior, one must study which of those can be replaced and how, in order to achieve it.

Software

There exists a large spectrum of cross-platform, open source implementations of deep learning methods which allow for the construction and training of AEs. This section summarizes the most popular frameworks, enumerates some specific implementations of AEs, and provides an example of use where an AE is implemented on top of one of these frameworks.

Available frameworks and packages

[153]: Abadi et al. (2016), "TensorFlow: A System for Large-scale Machine Learning"

Tensorflow [153] Developed by Google, Tensorflow has been the most influential deep learning framework. It is based on the concept of *data flow graphs*, where nodes represent mathematical operations and multidimensional data arrays travel through the edges. Its core is written in C++ and interfaces mainly with Python, although there are APIs for Java, C and Go as well.

[154]: Jia et al. (2014), "Caffe: Convolutional architecture for fast feature embedding"

Caffe [154] Originating at UC Berkeley, Caffe is built in C++ with speed and modularity in mind. Models are defined in a descriptive language instead of a common programming language, and trained in a C++ program.

[155]: Collobert et al. (2011), "Torch7: A Matlab-like Environment for Machine Learning"

Torch [155] It is a Lua library which promises speed and flexibility, but the most notorious feature is its large ecosystem of community-contributed tutorials and packages.

[156]: Chen et al. (2015), "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems"

MXNet [156] This project is currently held at the Apache Incubator for incoming projects into the Apache Foundation. It is written in C++ and Python, and offers APIs in several additional languages, such as R, Scala, Perl and Julia. MXNet provides flexibility in the definition of models, which can be programmed symbolically as well as imperatively.

[157]: Chollet et al. (2015), *Keras*

Keras [157] Keras is a higher-level library for deep learning in Python, and can rely on Tensorflow, Theano, MXNet or Cognitive Toolkit for the underlying operations. It simplifies the creation of deep learning architectures by providing several shortcuts and predefined utilities, as well as a common interface for several deep learning toolkits.

In addition to the previous ones, other well known deep learning frameworks are Theano [158], Microsoft Cognitive Toolkit (CNTK[‡]) and Chainer [§].

[158]: Theano Development Team (2016), "Theano: A Python framework for fast computation of mathematical expressions"

Setting various differences apart, all of these frameworks present some common traits when building AEs. Essentially, the user has to define the model layer by layer, placing activations where desired. When establishing the objective function, they will surely include

‡ `https://docs.microsoft.com/cognitive-toolkit/`
§ `https://chainer.org/`

the most usual ones, but uncommon loss functions such as correntropy or some regularizations such as contraction may need to be implemented additionally.

Very few pieces of software have specialized in the construction of AEs. Among them, there is an implementation of the sparse AE available in packages Autoencoder [159] and SAENET [160] of the CRAN repository for R, as well as an option for easily building basic AEs in H2O[¶]. The yadlt library for Python implements denoising AEs and several ways of stacking AEs.

[159]: Dubossarsky et al. (2015), *autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data*

[160]: Hogg et al. (2015), *SAENET: A Stacked Autoencoder Implementation with Interface to 'neuralnet'*

Example of use

For the purposes of the case study in Section 7, some simple implementations of different shallow AEs have been developed and published on a public code repository under a free software license[**]. In order to use these scripts, the machine will need to have Keras and Tensorflow installed. This can be achieved from a Python package manager, such as `pip` or `pipenv`, or even general package managers from some Linux distributions.

In the provided repository, the reader can find four scripts dedicated to AEs and one to PCA. Among the first ones, `autoencoder.py` defines the Keras model for a given AE type with the specified activation for the encoding layer. For its part, `utils.py` implements regularizations and modifications in order to be able to define basic, sparse, contractive, denoising and robust AEs.

Executable scripts are `mnist.py` and `cancer.py`. The first trains any AE with the MNIST dataset and outputs a graphical representation of the encoding and reconstruction of some test instances, whereas the latter needs the Wisconsin Breast Cancer Diagnosis (WDBC) dataset in order to train an AE for it. To use them, just call the Python interpreter with the script as an argument, e.g. `python mnist.py`.

In order to modify the learned model in one of these scripts, the user will need to adjust parameters in the construction of an `Autoencoder` object. The following is an example which will define a sparse denoising AE:

```
dsae = Autoencoder(
  input_dim    = 784,   encoding_dim = 36,
  weight_decay = False, sparse       = True,
```

[¶] http://docs.h2o.ai
https://deep-learning-tensorflow.readthedocs.io/
[**] https://github.com/fdavidcl/ae-review-resources

```
  contractive  = False, denoising    = True,
  robust       = False, activation   = "tanh"
)
```

Other numerical parameters for each AE type can be further customized inside the `build` method. The training process of this AE can be launched via a `MNISTTrainer` object:

```
MNISTTrainer(dsae).train(
  optimizer = "adam", epochs = 50,
  loss = losses.binary_crossentropy
).predict_test()
```

Finally, running the modified script will train the AE and output some graphical representations.

The `Autoencoder` class can be reused to train AEs with other datasets. For this, one would need to implement funtionality analogous to the `MNISTTrainer` class, which loads and prepares data, which is provided to the AE model to be trained. A different example can be found in the `CancerTrainer` class for the WDBC dataset.

Case study: handwritten digits

In order to offer some insight into the behavior of the main kinds of AE that can be applied to the same problem, as well as some of the key points in their configuration, we can study the resulting codifications and reconstructions when training them with the well known dataset of handwritten digits MNIST [38]. To do so, we have trained several AEs with the 60 000 training instances, and have obtained reconstructions for the first test instance of each class. Input values, originally ranging from 0 to 255, have been scaled to the $[0, 1]$ interval.

[38]: LeCun et al. (1998), "Gradient-based learning applied to document recognition"

By default, the architecture of every AE has been as follows: a 784-unit input layer, a 36-unit encoding layer with tanh activation and a 784-unit output layer with sigmoid activation. They have been trained with the RMSProp algorithm for a total of 60 epochs and use binary cross-entropy as their reconstruction error, except for the robust AE which uses its own loss function, correntropy. They are all provided identical weight initializations and hyperparameters.

Firstly, the performance impact of the encoding length and the optimizer is studied. Next, changes in the behavior of a standard AE due to different activation functions are analyzed. Lastly, the main AE models for feature fusion are compared sharing a common configuration. Scripts that were used to generate these results were implemented in Python, with the Keras library over the Tensorflow backend.

Settings of encoding length

As discussed previously, the number of units in the encoding layer can determine whether the AE is able to learn a useful representation. This fact is captured in Fig. I.13, where an encoding of 16 variables is too small for the shallow AE to be successfully trained with the default configuration, but a 36-variable codification achieves reasonably good reconstructions. The accuracy of these can be improved at the cost of enlarging the encodings, as can be seen with the 81 and 144-variable encodings. Square numbers were chosen for the encoding lengths for easier graphical representation, as will be seen in Section 7, but any other length would have been as valid.

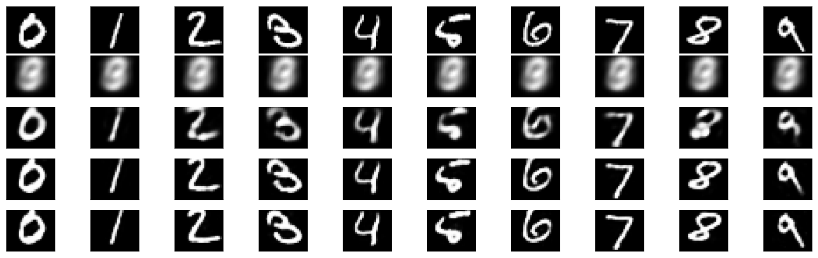


Figure I.13: First row: test samples; Remaining rows: reconstructions obtained with 16, 36, 81 and 144 units in the encoding layer, respectively.

Comparison of optimizers

As introduced in Section I.3, AEs can use several optimization methods, usually based on SGD. Each variant attempts to improve SGD in a different way, habitually by accumulating previous gradients in some way or dynamically adapting parameters such as the learning rate. Therefore, they will mainly differ in their ability to converge and their speed in doing so.

The optimizers used in these examples were baseline SGD, AdaGrad, Adam and RMSProp. Their progressive improvement of the objective function through the training phase is compared in Fig. I.14. It is easily observed that SGD variants vastly improve the basic method, and Adam obtains the best results among them, being closely followed by AdaGrad. The speed of convergence seems slightly higher in Adam as well.

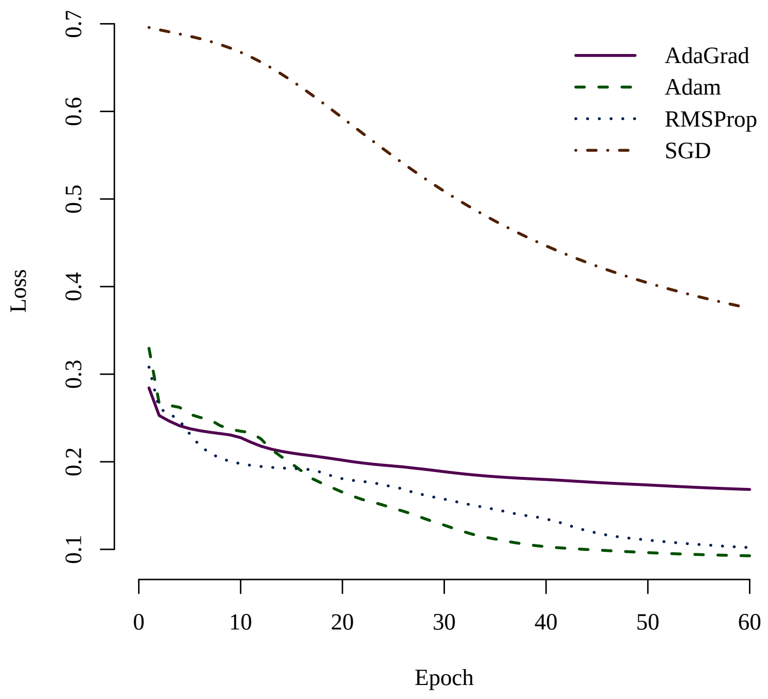


Figure I.14: Evolution of the loss function when using several optimizers.

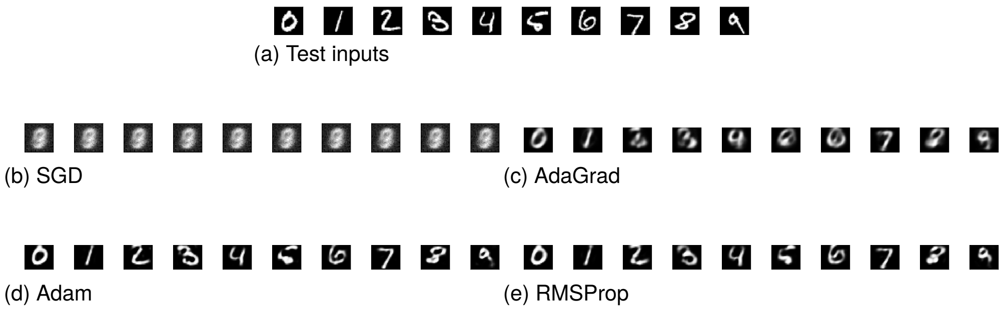


Figure I.15: Test samples and reconstructions obtained with different optimizers.

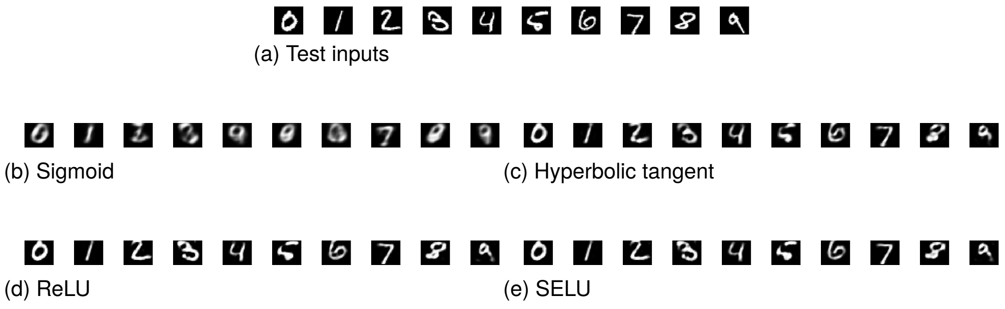
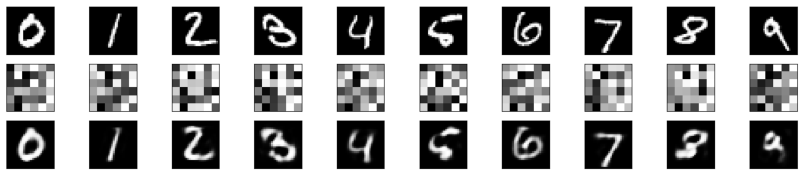
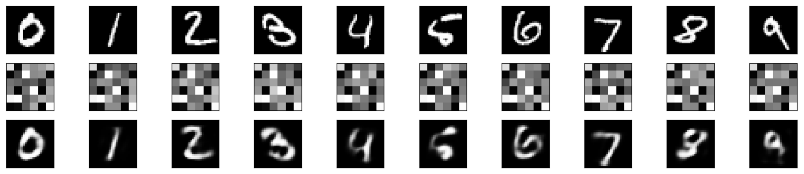
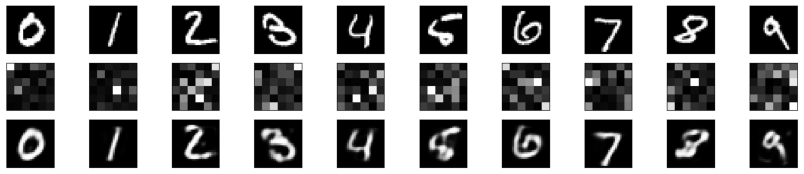
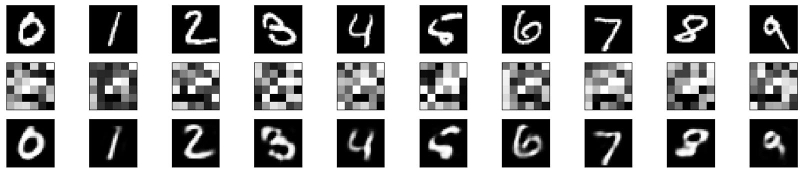


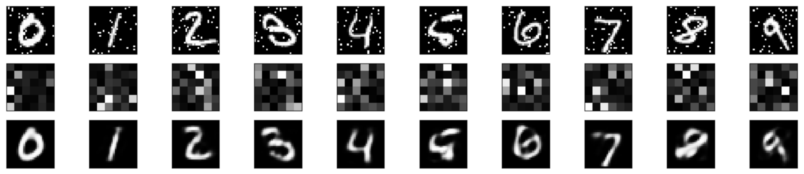
Figure I.16: Test samples and reconstructions obtained with different activation functions.


(a) Basic AE


(b) Basic AE with weight decay


(c) Sparse AE


(d) Contractive AE


(e) Denoising AE with sparsity regularization

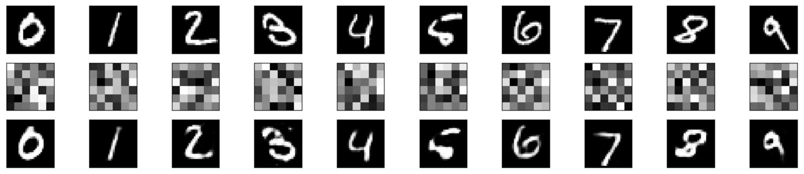

(f) Robust AE with weight decay

Figure I.17: Reconstructing test samples with different AE models. First row of each figure shows test samples, second row shows activations of the encoding layer and third row displays reconstructions. Encoded values range from -1 (black) to 1 (white).

In addition, Fig. I.15 provides the reconstructions generated for some test instances for a basic AE trained with each of those optimizers. As could be intuitively deduced by the convergence, or lack thereof, of the methods, SGD was not capable of finding weights which would recover any digit. AdaGrad, for its part, did improve on SGD but its reconstructions are relatively poor, whereas Adam and RMSProp display superior performance, with little difference between them.

Comparison of activation functions

Activation functions play an important role in the way gradients are propagated through the network. In this case, we apply four widely used activations in the encoding layer of a basic AE and compare how they affect its reconstruction ability. Some example results can be seen in Fig. I.16.

Sigmoid and hyperbolic tangent are functions with similar properties, but in spite of this they produce remarkably dissimilar results. Reconstructions are poor when using sigmoidal activation, while tanh achieves representations much closer to the original inputs.

With respect to ReLU and SELU, the observed results are surprisingly solid and almost indistinguishable. They perform slightly better than tanh in the sense that reconstructions are noticeably sharper. Their good performance in this case may be due to the nature of the data, which is restricted to the $[0, 1]$ interval and does not necessarily show the behavior of these activations in general.

Comparison of the main AE models

It can be interesting to study the different traits the codifications may acquire when variations on the basic AE are introduced. The reconstructions produced by six different AE models are shown in Fig. I.17.

The basic AE (Fig. I.17a) and the one with weight decay (Fig. I.17b) both generate recognizable reconstructions, although slighly blurry. They however do not produce much variability among different digits in the encoding layer, which means they are not making full use of its 36 dimensions. The weight decay corresponds to Eq. I.12 with λ set to 0.01.

The sparse AE has been trained according to Eq. I.15 with an expected activation value of -0.7. Its reconstructions are not much different from those of the previous ones, but in this case the encoding layer has much lower activations in average, as can be appreciated by the darker representations in Fig. I.17c. Most of the information is therefore tightly condensed in a few latent variables.

The contractive AE achieves other interesting properties in its encoding: it has attempted to model the data as a lower dimensional manifold, where digits that seem more similar will be placed closer than those which are very unalike. As a consequence, the 0 and the 1 shown in Fig. I.17d have very differing codifications, whereas the 3 and the 8 have relatively similar ones. Intuitively, one would need to travel larger distances along the learned manifold to go from a 0 to a 1, than from a 3 to an 8.

The denoising AE is able to eliminate noise from test instances, at the expense of losing some sharpness in the reconstruction, as can be seen in Fig. I.17e. Finally, the robust AE (Fig. I.17f) achieves noticeably higher clarity in the reconstruction and more variance in the encoding than the standard AEs.

I.7 Conclusions

As Pedro Domingos states in his famous tutorial [20], and as can be seen from the large number of publications on the subject, feature engineering is the key to obtain good machine learning models, able to generalize and provide decent performance. This process consists in choosing the most relevant subset of features or combining some of them to create new ones. Automated fusion of features, specially when performed by nonlinear techniques, has demonstrated to be very effective. Neural network-based autoencoders are among the approaches to conduct this kind of task.

[20]: Domingos (2012), "A few useful things to know about machine learning"

This paper started offering the reader with a general view of which an AE is, as well as its essential foundations. After introducing the usual AE network structures, a new AE taxonomy, based on the properties of the inferred model, has been proposed. Those AE models mainly used in feature fusion have been explained in detail, highlighting their most salient characteristics and comparing them with more classical feature fusion techniques. The use of disparate activation functions and training methods for AEs has been also thoroughly illustrated.

In addition to AEs for feature fusion, many other AE models and applications have been listed. The number of new proposals in this field is always growing, so it is easy to find dozens of AE variants, most of them based on the fundamental models described above.

This review is complemented by a final section proposing guidelines for selecting the most appropriate AE model based on different criteria, such as the type of units, loss function, activation function, etc., as well as mentioning available software to put this knowledge into practice. Empirical results on the well known MNIST dataset obtained from several AE configurations, combining disparate activation functions, optimizers and models, have been compared. The aim is to offer the reader help when facing this type of decision.

Acknowledgments: This work is supported by the Spanish National Research Projects TIN2015-68454-R and TIN2014-57251-P, and Project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

I.A. Description of used datasets

Breast Cancer Diagnosis (Wisconsin)

[139]: Mangasarian et al. (1995), "Breast cancer diagnosis and prognosis via linear programming"

The well known dataset of diagnosis of breast cancer in Wisconsin (WDBC) [139] is briefly used in Section I.5 to provide a 2-dimensional visualization example.

This dataset consists of 569 instances corresponding to patients, each of which present 30 numeric input features and one of two classes that identify the type of tumor: benign or malignant. The dataset is slightly imbalanced, exhibiting a 37.3% of instances associated to the malignant class, while the remaining 62.7% correspond to benign tumors. The data have been normalized for the training process of the basic AE that generated the example.

Originally, features were extracted from a digitized image of a fine-needle aspiration sample of a breast mass, and described ten different traits of each cell nucleus. The mean, standard error and largest value of these features are computed, resulting in the 30 input attributes for each patient, gathered in the published dataset.

WDBC is usually relied on as an example dataset and most classifiers generally obtain high accuracy: the authors of the original proposal already achieved 97% of classification accuracy in cross-validation. However, it presents some issues when applying AEs: its small imbalance may cause instances classified as benign to contribute more to the loss function, inducing some bias in the resulting network, which may reconstruct these more accurately than the rest. Furthermore, it is composed of relatively few instances, which may not be sufficient for some deep learning techniques to be able to generalize.

MNIST

MNIST [38] is a widely used dataset within deep learning research. It is regularly chosen as a benchmark for new techniques and neural architectures. It has been the base of our case study in Section 7.

[38]: LeCun et al. (1998), "Gradient-based learning applied to document recognition"

The dataset consists of 60 000 instances, divided into a 50 000-instance set for training and the remaining 10 000 for test. each corresponding to a 28x28-sized image of a handwritten digit, from 0 to 9. The values of this 28x28 matrix or 784-variable input represent the gray level of each pixel, and therefore range from 0 to 255, but they have been rescaled to the $[0, 1]$ interval in our examples.

This dataset is actually a modified subset of a previous work from NIST[††] for character recognition. The original images used only black or white pixels, whereas in MNIST they have been anti-aliased.

MNIST has been used as benchmark for a large variety of deep learning proposals, since it is reasonably easy to extract higher-level features out of simple grayscale images, and it provides a high enough amount of training data. State-of-the-art work[‡‡] achieves an error rate of around 0.2% [120, 161].

[120]: Alvear-Sandoval et al. (2018), "On building ensembles of stacked denoising auto-encoding classifiers and their further improvement"
[161]: Ciregan et al. (2012), "Multi-column deep neural networks for image classification"

†† Available at `http://doi.org/10.18434/T4H01C`.
‡‡ A collection of methods applied to MNIST and their results is available at `http://yann.lecun.com/exdb/mnist/`.

References

[1] David Charte, Francisco Charte, Salvador García, María José del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007.

[2] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/BF02478259.

[3] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. John Wiley and Sons, 1949.

[4] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton (Project PARA)*. Cornell Aeronautical Laboratory, 1957.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–538. DOI: 10.1038/323533a0.

[6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[7] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[8] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527.

[10] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural networks : the official journal of the International Neural Network Society* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[11] Geoffrey E Hinton. “Deep belief networks”. In: *Scholarpedia* 4.5 (2009), p. 5947. DOI: 10.4249/scholarpedia.5947.

[12] Yann LeCun and Yoshua Bengio. “The Handbook of Brain Theory and Neural Networks”. In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258.

[13] Ronald J Williams and David Zipser. “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural computation* 1.2 (1989), pp. 270–280. DOI: 10.1162/neco.1989.1.2.270.

[14] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[15] Marc'Aurelio Ranzato, Y-Lan Boureau, Sumit Chopra, and Yann LeCun. "A Unified Energy-Based Framework for Unsupervised Learning". In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by Marina Meila and Xiaotong Shen. Vol. 2. Proceedings of Machine Learning Research. San Juan, Puerto Rico: PMLR, Mar. 2007, pp. 371–379.

[16] Dana H. Ballard. "Modular Learning in Neural Networks". In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*. AAAI'87. Seattle, Washington: AAAI Press, 1987, pp. 279–284.

[17] R Bellman. *Dynamic Programming*. Princeton University Press, 1957, p. 342.

[18] Manoranjan Dash and Huan Liu. "Feature Selection for Classification". In: *Intelligent Data Analysis* 1 (1997), pp. 131–156. DOI: 10.3233/IDA-1997-1302.

[19] Huan Liu and Hiroshi Motoda. *Feature extraction, construction and selection: A data mining perspective*. Vol. 453. Springer Science & Business Media, 1998.

[20] Pedro Domingos. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), pp. 78–87. DOI: 10.1145/2347736.2347755.

[21] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50. (Visited on 09/05/2017).

[22] Geoffrey E Hinton. "Learning distributed representations of concepts". In: *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. Amherst, MA. 1986, p. 12. DOI: 10.1109/69.917563.

[23] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015. Chap. 7, pp. 163–194.

[24] Dietrich Wettschereck, David W Aha, and Takao Mohri. "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms". In: *Lazy learning*. Springer, 1997, pp. 273–314. DOI: 10.1023/A:1006593614256.

[25] Mark Andrew Hall. "Correlation-based feature selection for machine learning". PhD thesis. University of Waikato Hamilton, 1999.

[26] Hanchuan Peng, Fuhui Long, and Chris H. Q. Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2005), pp. 1226–1238. DOI: 10.1109/TPAMI.2005.159.

[27] Pabitra Mitra, C. A. Murthy, and Sankar K. Pal. "Unsupervised Feature Selection Using Feature Similarity". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.3 (2002), pp. 301–312. DOI: 10.1109/34.990133.

[28] Salvador García, Julián Luengo, and Francisco Herrera. "Tutorial on practical tips of the most influential data preprocessing algorithms in data mining". In: *Knowledge-Based Systems* 98 (2016), pp. 1–29. DOI: 10.1016/j.knosys.2015.12.006.

[29] Isabelle Guyon and André Elisseeff. "An Introduction to Feature Extraction". In: *Feature Extraction: Foundations and Applications*. Ed. by Isabelle Guyon, Masoud Nikravesh, Steve Gunn, and Lotfi A. Zadeh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–25. DOI: 10.1007/978-3-540-35488-8_1.

[30] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 6* 2.11 (Nov. 1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[31] Harold Hotelling. "Analysis of a complex of statistical variables into principal components". In: *Journal of educational psychology* 24.6 (1933), p. 417. DOI: 10.1037/h0071325.

[32] Ronald A Fisher. "The statistical utilization of multiple measurements". In: *Annals of Human Genetics* 8.4 (1938), pp. 376–386. DOI: 10.1111/j.1469-1809.1938.tb02189.x.

[33] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural computation* 10.5 (1998), pp. 1299–1319. DOI: 10.1162/089976698300017467.

[34] Joshua B Tenenbaum, Vin de Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.

[35] Lawrence Cayton. *Algorithms for manifold learning*. Tech. rep. University of California at San Diego, 2005, pp. 1–17.

[36] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[37] Utthara Gosa Mangai, Suranjana Samanta, Sukhendu Das, and Pinaki Roy Chowdhury. "A survey of decision fusion and feature fusion strategies for pattern classification". In: *IETE Technical review* 27.4 (2010), pp. 293–307. DOI: 10.4103/0256-4602.64604.

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[39] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243. DOI: 10.1002/aic.690370209.

[40] Holger Schwenk and Yoshua Bengio. "Training methods for adaptive boosting of neural networks". In: *Advances in neural information processing systems*. 1998, pp. 647–653. DOI: 10.1162/089976600300015178.

[41] Robert Hecht-Nielsen. "Replicator neural networks for universal optimal source coding". In: *Science* (1995), pp. 1860–1863. DOI: 10.1126/science.269.5232.1860.

[42] Davide Chicco, Peter Sadowski, and Pierre Baldi. "Deep autoencoder neural networks for gene ontology annotation predictions". In: *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM. 2014, pp. 533–540. DOI: 10.1145/2649387.2649442.

[43] Pierre Baldi and Kurt Hornik. "Neural networks and principal component analysis: Learning from examples without local minima". In: *Neural networks* 2.1 (1989), pp. 53–58. DOI: 10.1016/0893-6080(89)90014-2.

[44] Hanna Kamyshanska and Roland Memisevic. “On autoencoder scoring”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 720–728.

[45] Li Deng et al. “Binary coding of speech spectrograms using a deep auto-encoder”. In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010.

[46] Pierre Baldi. “Autoencoders, unsupervised learning, and deep architectures”. In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 37–49. (Visited on 08/28/2017).

[47] Donald E Knuth. “Two notes on notation”. In: *The American Mathematical Monthly* 99.5 (1992), pp. 403–422. DOI: 10.2307/2325085.

[48] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Domain adaptation for large-scale sentiment classification: A deep learning approach”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 513–520.

[49] Çağlar Gülçehre and Yoshua Bengio. “Knowledge matters: Importance of prior information for optimization”. In: *Journal of Machine Learning Research* 17.8 (2016), pp. 1–32.

[50] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. “Self-Normalizing Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[51] Oleksii Kuchaiev and Boris Ginsburg. “Training Deep AutoEncoders for Recommender Systems”. In: *International Conference on Learning Representations*. 2018.

[52] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus -Robert Müller. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. DOI: 10.1007/3-540-49430-8_2.

[53] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. “Exploring strategies for training deep neural networks”. In: *Journal of Machine Learning Research* 10.Jan (2009), pp. 1–40.

[54] Hervé Bourlard and Yves Kamp. “Auto-association by multilayer perceptrons and singular value decomposition”. In: *Biological cybernetics* 59.4 (1988), pp. 291–294. DOI: 10.1007/BF00332918.

[55] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. “Greedy layer-wise training of deep networks”. In: *Advances in neural information processing systems*. 2007, pp. 153–160.

[56] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The annals of mathematical statistics* (1951), pp. 400–407. DOI: 10.1007/978-1-4612-5110-1_9.

[57] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[58] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-RMSProp”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[59] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations*. 2015.

[60] Jiquan Ngiam et al. “On optimization methods for deep learning”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 265–272.

[61] Augustin Cauchy. “Méthode générale pour la résolution des systemes d’équations simultanées”. In: *Comptes Rendus des Séances de l’Académie des Sciences* A.25 (1847), pp. 536–538.

[62] Anders Krogh and John A Hertz. “A simple weight decay can improve generalization”. In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[63] Bruno A Olshausen and David J Field. “Sparse coding with an overcomplete basis set: A strategy employed by V1?” In: *Vision research* 37.23 (1997), pp. 3311–3325. DOI: 10.1016/S0042-6989(97)00169-7.

[64] Bruno A Olshausen and David J Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. In: *Nature* 381.6583 (1996), pp. 607–609. DOI: 10.1038/381607a0.

[65] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. “Sparse deep belief net model for visual area V2”. In: *Advances in neural information processing systems*. 2008, pp. 873–880.

[66] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[67] Andrew Ng. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[68] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. “Efficient learning of sparse representations with an energy-based model”. In: *Advances in neural information processing systems*. 2007, pp. 1137–1144.

[69] Y-lan Boureau, Yann L Cun, et al. “Sparse feature learning for deep belief networks”. In: *Advances in neural information processing systems*. 2008, pp. 1185–1192.

[70] Salah Rifai, Yoshua Bengio, Pascal Vincent, and Yann N Dauphin. “A Generative Process for sampling Contractive Auto-Encoders”. In: *Proceedings of the 29th International Coference on International Conference on Machine Learning*. ICML’12. Edinburgh, Scotland: Omnipress, 2012, pp. 1811–1818.

[71] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. “Extracting and Composing Robust Features with Denoising Autoencoders”. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. ACM, 2008, pp. 1096–1103.

[72] Yann LeCun. “Modèles connexionnistes de l’apprentissage”. PhD thesis. These de Doctorat, Université Paris 6, 1987.

[73] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion”. In: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408.

[74] Yu Qi, Yueming Wang, Xiaoxiang Zheng, and Zhaohui Wu. “Robust feature learning by stacked autoencoder with maximum correntropy criterion”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6716–6720. DOI: 10.1109/ICASSP.2014.6854900.

[75] Weifeng Liu, Puskal P. Pokharel, and Jose C. Principe. “Correntropy: A localized similarity measure”. In: *IEEE International Joint Conference on Neural Networks, 2006. IJCNN*. IEEE, 2006, pp. 4919–4924. DOI: 10.1109/IJCNN.2006.247192.

[76] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction". In: *Artificial Neural Networks and Machine Learning – ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I*. Ed. by Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski. Springer Berlin Heidelberg, 2011, pp. 52–59. DOI: 10.1007/978-3-642-21735-7_7.

[77] Yann LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[78] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. "Unsupervised learning of video representations using LSTMs". In: *International Conference on Machine Learning*. 2015, pp. 843–852.

[79] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[80] Diederik P. Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013). (Visited on 08/28/2017).

[81] Charles W Fox and Stephen J Roberts. "A tutorial on variational Bayesian inference". In: *Artificial intelligence review* (2012), pp. 1–11. DOI: 10.1007/s10462-011-9236-8.

[82] Alexey Dosovitskiy and Thomas Brox. "Generating images with perceptual similarity metrics based on deep networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 658–666.

[83] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014, pp. 1278–1286.

[84] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).

[85] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[86] Otto Fabius and Joost R van Amersfoort. "Variational recurrent auto-encoders". In: *International Conference on Learning Representations*. 2015.

[87] Alireza Makhzani and Brendan J Frey. "PixelGAN Autoencoders". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 1972–1982. (Visited on 12/08/2017).

[88] Yuchen Pu et al. "Adversarial symmetric variational autoencoder". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4331–4340.

[89] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J. Kennedy. "Relational autoencoder for feature extraction". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 364–371. DOI: 10.1109/IJCNN.2017.7965877.

[90] Sebastien Razakarivony and Frédéric Jurie. "Discriminative autoencoders for small targets detection". In: *22nd IEEE International Conference on Pattern Recognition (ICPR)*. IEEE, 2014, pp. 3528–3533. DOI: 10.1109/ICPR.2014.607. (Visited on 08/28/2017).

[91] Ozan İrsoy and Ethem Alpaydın. "Unsupervised feature extraction with autoencoder trees". In: *Neurocomputing* 258 (Oct. 2017), pp. 63–73. DOI: 10.1016/j.neucom.2017.02.075. (Visited on 08/28/2017).

[92] Fuzhen Zhuang et al. "Representation learning via Dual-Autoencoder for recommendation". In: *Neural Networks* 90 (June 2017), pp. 83–89. DOI: 10.1016/j.neunet.2017.03.009.

[93] Roland Memisevic. "Gradient-based learning of higher-order image features". In: *IEEE International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 1591–1598. DOI: 10.1109/ICCV.2011.6126419.

[94] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. "Semi-supervised recursive autoencoders for predicting sentiment distributions". In: *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2011, pp. 151–161.

[95] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection". In: *Advances in Neural Information Processing Systems*. 2011, pp. 801–809.

[96] Fei Wang and Jimeng Sun. "Survey on distance metric learning and dimensionality reduction in data mining". In: *Data Mining and Knowledge Discovery* 29.2 (2015), pp. 534–564. DOI: 10.1007/s10618-014-0356-z.

[97] Ruslan Salakhutdinov and Geoffrey E Hinton. "Learning a nonlinear embedding by preserving class neighbourhood structure". In: *International Conference on Artificial Intelligence and Statistics*. 2007, pp. 412–419.

[98] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. *Dimensionality reduction: a comparative review*. Tech. rep. 2009.

[99] Ian T Jolliffe. *Principal component analysis*. Springer, 1986.

[100] Ian T Jolliffe. "Principal Component Analysis and Factor Analysis". In: *Principal component analysis*. Springer, 1986, pp. 115–128. DOI: 10.1007/978-1-4757-1904-8.

[101] Ronald A. Fisher. "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS". In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[102] Warren S Torgerson. "Multidimensional scaling: I. Theory and method". In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916.

[103] John W Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on computers* 100.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678.

[104] Wenchao Yu et al. "Embedding with autoencoder regularization". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 208–223. DOI: 10.1007/978-3-642-40994-3_14.

[105] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[106] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation". In: *Neural Computation* 15 (2003), pp. 1373–1396. DOI: 10.1162/089976603321780317.

[107] Kui Jia, Lin Sun, Shenghua Gao, Zhan Song, and Bertram E Shi. "Laplacian Auto-Encoders: An explicit learning of nonlinear data manifold". In: *Neurocomputing* 160 (2015), pp. 250–260. DOI: 10.1016/j.neucom.2015.02.023.

[108] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. Deep generative models, pp. 651–716.

[109] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. Colorado University at Boulder, Department of Computer Science, 1986.

[110] Philip M Long and Rocco Servedio. "Restricted Boltzmann machines are hard to approximately evaluate or simulate". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 703–710.

[111] Geoffrey E. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: *Neural Computation* 14.8 (Aug. 2002), pp. 1771–1800. DOI: 10.1162/089976602760128018.

[112] Jun Xu et al. "Stacked Sparse Autoencoder (SSAE) for Nuclei Detection on Breast Cancer Histopathology Images". In: *IEEE Transactions on Medical Imaging* 35.1 (Jan. 2016), pp. 119–130. DOI: 10.1109/tmi.2015.2458702.

[113] Chang Xu, Dacheng Tao, and Chao Xu. "A survey on multi-view learning". In: *arXiv preprint arXiv:1304.5634* (2013).

[114] TengQi Ye, Tianchun Wang, Kevin McGuinness, Yu Guo, and Cathal Gurrin. "Learning Multiple Views with Orthogonal Denoising Autoencoders". In: *MultiMedia Modeling*. Ed. by Qi Tian et al. Vol. 9516. Cham: Springer International Publishing, 2016, pp. 313–324. DOI: 10.1007/978-3-319-27671-7_26. (Visited on 09/06/2017).

[115] Yanan Liu, Xiaoqing Feng, and Zhiguang Zhou. "Multimodal video classification with stacked contractive autoencoders". In: *Signal Processing* 120 (Mar. 2016), pp. 761–766. DOI: 10.1016/j.sigpro.2015.01.001. (Visited on 08/28/2017).

[116] Francisco Herrera, Francisco Charte, Antonio J. Rivera, and María J. del Jesus. *Multilabel Classification. Problem analysis, metrics and techniques*. Springer, 2016.

[117] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. "Learning Deep Latent Space for Multi-Label Classification". In: *AAAI Conference on Artificial Intelligence*. 2017.

[118] Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. "Variational Autoencoder for Semi-Supervised Text Classification". In: *AAAI Conference on Artificial Intelligence*. 2017.

[119] Nal Kalchbrenner and Phil Blunsom. "Recurrent Continuous Translation Models." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Vol. 3. 39. 2013, p. 413.

[120] Ricardo F. Alvear-Sandoval and Aníbal R. Figueiras-Vidal. "On building ensembles of stacked denoising auto-encoding classifiers and their further improvement". In: *Information Fusion* 39 (Jan. 2018), pp. 41–52. DOI: 10.1016/j.inffus.2017.03.008.

[121] Gregory K Wallace. "The JPEG still picture compression standard". In: *IEEE transactions on consumer electronics* 38.1 (1992), pp. xviii–xxxiv. DOI: 10.1145/103085.103089.

[122] Chun Chet Tan and Chikkannan Eswaran. "Using Autoencoders for Mammogram Compression". In: *Journal of Medical Systems* 35.1 (Feb. 2011), pp. 49–58. DOI: 10.1007/s10916-009-9340-3. (Visited on 08/28/2017).

[123] Thierry Dumas, Aline Roumy, and Christine Guillemot. "Image Compression with Stochastic Winner-Take-All Auto-Encoder". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017)*. 2017. DOI: 10.1109/ICASSP.2017.7952409.

[124] Davide Del Testa and Michele Rossi. "Lightweight lossy compression of biometric patterns via denoising autoencoders". In: *IEEE Signal Processing Letters* 22.12 (2015), pp. 2304–2308. DOI: 10.1109/LSP.2015.2476667.

[125] Y Miao and P Blunsom. "Language as a latent variable: Discrete generative models for sentence compression". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 319–328.

[126] Daniel Hsu. "Time Series Compression Based on Adaptive Piecewise Recurrent Autoencoder". In: *arXiv preprint arXiv:1707.07961* (2017).

[127] Charu C Aggarwal. "An introduction to outlier analysis". In: *Outlier analysis*. Springer, 2013, pp. 1–40. DOI: 978-3-319-47578-3_1.

[128] Mayu Sakurada and Takehisa Yairi. "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction". In: ACM Press, 2014, pp. 4–11. DOI: 10.1145/2689746.2689747. (Visited on 08/28/2017).

[129] Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. "Outlier Detection with Autoencoder Ensembles". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM. 2017, pp. 90–98. DOI: 10.1137/1.9781611974973.11.

[130] Jacopo Castellini, Valentina Poggioni, and Giulia Sorbi. "Fake Twitter followers detection by denoising autoencoder". In: *Proceedings of the International Conference on Web Intelligence - WI '17*. New York, New York, USA: ACM Press, 2017, pp. 195–202. DOI: 10.1145/3106426.3106489.

[131] Lianhua Chi and Xingquan Zhu. "Hashing Techniques: A Survey and Taxonomy". In: *ACM Computing Surveys (CSUR)* 50.1 (2017), p. 11. DOI: 10.1145/3047307.

[132] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. "Similarity Search in High Dimensions via Hashing". In: *Proceedings of the 25th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc. 1999, pp. 518–529.

[133] Ruslan Salakhutdinov and Geoffrey Hinton. "Semantic hashing". In: *International Journal of Approximate Reasoning* 50.7 (2009), pp. 969–978. DOI: 10.1016/j.ijar.2008.11.006.

[134] Gerard Salton, Edward A Fox, and Harry Wu. "Extended Boolean information retrieval". In: *Communications of the ACM* 26.11 (1983), pp. 1022–1036. DOI: 10.1145/182.358466.

[135] Miguel Carreira-Perpinan and Weiran Wang. "Distributed optimization of deeply nested systems". In: *Artificial Intelligence and Statistics*. 2014, pp. 10–19.

[136] Miguel A. Carreira-Perpinán and Ramin Raziperchikolaei. "Hashing with binary autoencoders". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 557–566. DOI: 10.1109/CVPR.2015.7298654. (Visited on 08/28/2017).

[137] Usama M Fayyad, Andreas Wierse, and Georges G Grinstein. *Information visualization in data mining and knowledge discovery*. Morgan Kaufmann, 2002.

[138] G. E. Hinton. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (July 28, 2006), pp. 504–507. DOI: 10.1126/science.1127647. (Visited on 08/26/2017).

[139] Olvi L Mangasarian, W Nick Street, and William H Wolberg. "Breast cancer diagnosis and prognosis via linear programming". In: *Operations Research* 43.4 (1995), pp. 570–577. DOI: 10.1287/opre.43.4.570.

[140] Ting-Chung Poon. *Digital holography and three-dimensional display: Principles and Applications*. Springer Science & Business Media, 2006.

[141] Tomoyoshi Shimobaba et al. "Autoencoder-based holographic image restoration". In: *Applied Optics* 56.13 (Feb. 2017), F27. DOI: 10.1364/ao.56.000f27.

[142] Lovedeep Gondara. "Medical Image Denoising Using Convolutional Denoising Autoencoders". In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)* (2016), pp. 241–246. DOI: 10.1109/ICDMW.2016.0041.

[143] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. "Speech enhancement based on deep denoising autoencoder". In: *14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. 2013.

[144] Takaaki Ishii, Hiroki Komiyama, Takahiro Shinozaki, Yasuo Horiuchi, and Shingo Kuroiwa. "Reverberant speech recognition based on denoising autoencoder". In: *14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. 2013.

[145] D T Grozdić and Slobodan T Jovičić. "Whispered Speech Recognition Using Deep Denoising Autoencoder and Inverse Filtering". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.12 (2017), pp. 2313–2322. DOI: 10.1109/TASLP.2017.2738559.

[146] Gene H Golub and Christian Reinsch. "Singular value decomposition and least squares solutions". In: *Numerische mathematik* 14.5 (1970), pp. 403–420. DOI: 10.1007/BF02163027.

[147] Chaoqun Hong, Jun Yu, Jian Wan, Dacheng Tao, and Meng Wang. "Multimodal Deep Autoencoder for Human Pose Recovery". In: *IEEE Transactions on Image Processing* 24.12 (Dec. 2015), pp. 5659–5670. DOI: 10.1109/tip.2015.2487860.

[148] Haoyin Zhou, Tao Zhang, and Weining Lu. "Vision-based pose estimation from points with unknown correspondences". In: *IEEE Transactions on Image Processing* 23.8 (2014), pp. 3468–3477. DOI: 10.1109/TIP.2014.2329765.

[149] Jesse Vig, Shilad Sen, and John Riedl. "Tagsplanations: explaining recommendations using tags". In: *Proceedings of the 14th international conference on Intelligent user interfaces*. ACM. 2009, pp. 47–56. DOI: 10.1145/1502650.1502661.

[150] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. "QUINTA: a question tagging assistant to improve the answering ratio in electronic forums". In: *IEEE EUROCON 2015-International Conference on Computer as a Tool*. IEEE. 2015, pp. 1–6. DOI: 10.1109/EUROCON.2015.7313677.

[151] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. "Relational Stacked Denoising Autoencoder for Tag Recommendation". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[152] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. "A survey on deep learning for big data". In: *Information Fusion* 42 (2018), pp. 146–157. DOI: 10.1016/j.inffus.2017.10.006.

[153] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, et al. “TensorFlow: A System for Large-scale Machine Learning”. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI’16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[154] Yangqing Jia et al. “Caffe: Convolutional architecture for fast feature embedding”. In: (2014), pp. 675–678.

[155] R. Collobert, K. Kavukcuoglu, and C. Farabet. “Torch7: A Matlab-like Environment for Machine Learning”. In: *BigLearn, NeurIPS Workshop*. 2011.

[156] Tianqi Chen et al. “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems”. In: *arXiv preprint arXiv:1512.01274* (2015).

[157] François Chollet et al. *Keras*. `https://github.com/fchollet/keras`. 2015.

[158] Theano Development Team. “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688 (May 2016).

[159] Eugene Dubossarsky and Yuriy Tyshetskiy. *autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data*. R package version 1.1. 2015.

[160] Stephen Hogg and Eugene Dubossarsky. *SAENET: A Stacked Autoencoder Implementation with Interface to 'neuralnet'*. R package version 1.1. 2015.

[161] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. “Multi-column deep neural networks for image classification”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2012, pp. 3642–3649.

Ruta: implementations of neural autoencoders in R

II

Source

David Charte, Francisco Herrera, and Francisco Charte. "Ruta: Implementations of neural autoencoders in R". In: *Knowledge-Based Systems* 174 (2019), pp. 4–8. DOI: 10.1016/j.knosys.2019.01.014

Abstract

Autoencoders are neural networks which perform feature learning on data. Many variants can be found in the literature, but their implementations are scarce, in separate software pieces and utilizing different languages and frameworks. The `ruta` package implements a unified foundation for the construction and training of autoencoders on top of Keras and Tensorflow, and allows for easy access to the main functionalities as well as full customization of their diverse aspects.

Keywords

unsupervised learning - neural networks - autoencoders

II.1 Introduction

The problem of feature extraction consists in finding a transformation of the feature space of some data set which is more adequate than the original one in relation to another task, such as classification or visualization. A particular case of this problem is dimensionality reduction, where the objective is to build a more compact representation for the data while retaining most of their information.

Some traditional techniques for feature extraction are principal components analysis (PCA) [2], multidimensional scaling [3], Isomap [4] and locally linear embedding [5]. Other more modern methods

[2]: Jolliffe (1986), *Principal component analysis*

[3]: Torgerson (1952), "Multidimensional scaling: I. Theory and method"

[4]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

[5]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

[6]: Maaten et al. (2008), "Visualizing data using t-SNE"

include t-distributed stochastic neighbor embedding (t-SNE) [6], which is designed to visualize high-dimensional datasets, restricted Boltzmann machines (RBMs) [7] and autoencoders (AEs) [8], both based on neural networks.

[7]: Goodfellow et al. (2016), "Deep Learning"

[8]: Hinton (2006), "Reducing the Dimensionality of Data with Neural Networks"

AEs are a tool for feature extraction in increasing development. Making use of them, however, is not straightforward. Software pieces which implement them are uncommon and are either very basic versions or adapted to specific databases. Basic AE models are relatively easy to implement in well-known deep learning frameworks, such as Keras [9] or Tensorflow [10], but this requires some knowledge about their structure and training procedures. In addition to this, some useful regularizations and alterations in the objective functions can present challenges while coding. Since most neural AEs share a common basis, it is desirable to have an implementation which abstracts its components and gives the customization possibilities to build different kinds of AEs without reimplementing them. This would allow users to leverage the possibilities of AEs as feature learning techniques without the need to study their architecture in advance.

[9]: Chollet et al. (2015), *Keras*

[10]: Abadi et al. (2016), "TensorFlow: A System for Large-scale Machine Learning"

The `ruta` package for the R language includes all the necessary foundations to build AEs for all kinds of experimentations. It is based on frameworks Keras and Tensorflow to ensure efficiency and cross-platform compatibility. Its interface allows any R user to easily define different models, train them and perform additional tasks with little to no previous knowledge required.

II.2 Problems and Background

As previously stated, the main objective of an AE is to find a good transformation of the features according to one or more criteria. When an instance is mapped to the new feature space, it is seen as an *encoding* of the original. This encoding must allow the AE to reconstruct the instance from the original feature space by means of a decodification process. Intuitively, this reconstruction can only be achieved if sufficient information about each instance is retained within the encoding.

Autoencoder framework

[11]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

An AE [11] is an artificial neural network (ANN) composed of an encoder and a decoder. Analytically, it can be seen as a composition of maps f and g which results in a tensor of the same shape as the input. As an ANN, it takes a form analogue to that on Fig. II.1. AEs were originally used to perform a preliminary weight training on other ANNs, but on their own they can also learn alternative representations for input data.

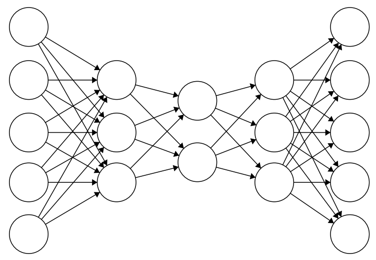


Figure II.1: A possible neural architecture for an AE with a 2-variable encoding layer.

The different aspects that lead an AE to a specific transformation are its neural architecture, which determines the type of input and the size of the encoding; the cost and activation functions, which can be defined and regularized in order to induce some desired properties, and parameters of the training process, such as the optimization algorithm or the number of times the data is feeded to the network.

Variants

An interesting advantage of AEs is their versatility: one can obtain encodings with certain properties if the adequate regularizations are chosen. There exist many AE variants in the literature [11], the most common ones centered in how to control the behavior of the transformation while allowing for faithful reconstructions. The following are the most relevant ones:

[11]: Charte et al. (2018), “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”

- Sparse: induces a low number of activations in average in the encoding layer.
- Contractive: attempts to preserve the local structure of the original space, thus searching for coordinates in a lower-dimensional manifold.
- Denoising: is able to remove noise introduced in input examples.
- Robust: is less sensitive to noise in instances due to a different loss function.
- Variational: extracts a generative model from the data and is able to produce new, unseen instances.
- Adversarial: trains in an adversarial manner with the aim of forcing the encoding to follow a given distribution.
- Convolutional and LSTM-based: are composed of other types of units and layers in order to accomodate bidimensional and sequential data, respectively.

II.3 Software Framework

In this section we elaborate on the internal structure of the developed software and its functionality.

Software Architecture

The object system utilized in `ruta` is S3, a minimal object orientation from the R language based on generic functions. The software is developed around several classes which have certain applicable methods:

- `ruta_autoencoder`: represents a parametrized AE learner. It can be trained and can perform several post-train tasks, such as data encoding and reconstruction.
- `ruta_network`: defines neural network structures by layers. Networks can be concatenated to produce a longer one.
- `ruta_loss`: represents the loss function to be optimized by the learner. It is either a wrapper over a loss function from Keras, or a built-in loss function such as correntropy.
- `ruta_noise`: represents a type of noise which can be applied to input data. Several of these are provided within the package for convenience.

Software Functionalities

The main functionalities of package `ruta` are as follows:

- Define and customize diverse aspects of an AE model.
- Train AE variants according to the desired objective function.
- Encode and reconstruct input data with a trained model.
- Evaluate a trained model according to several metrics which account for quality of reconstruction.
- Sample generative models created by variational AEs.
- Generate corrupted data with different types of noise.

The programming interface provided by the package gives several ways to access this set of functionalities, according to the desired level of customization and difficulty:

- Directly train an AE and compress a database via function `autoencode`.
- Define a basic AE simply by enumerating the dimensions of its layers in a vector, e.g. `autoencoder(c(32, 6))`.
- Define each layer composing the neural architecture by means of functions `input`, `dense`, `conv`, `output`, etc., then construct an AE with possibly one or more variant properties.

The following AE types can be used: basic, sparse, contractive, denoising, robust, variational and convolutional (via the included `conv` layers). Some of them may be combined by means of the `make` family of functions, e.g. `make_sparse`. They are extensively documented within the package and in the online documentation*.

Implementation Details

Since `ruta` is implemented on top of Tensorflow and Keras, it can run on computing devices such as GPUs. In order for them to be used, the correct Tensorflow version with CUDA support will need to be installed. Several issues can arise during the installation and first use, which have been documented in the troubleshooting section of the online documentation.

Few other software pieces provide the necessary functionality to build custom AEs. Among them we can find H2O [12], with its `h2o.deeplearning` function which includes an `autoencoder` option; package `autoencoder` for R [13], and library `yadlt` for Python. These focus on just one or two AE variants and provide less customizability than AEs defined in `ruta`. For further options one needs to resort to Deep Learning frameworks, which require a much higher programming effort in order to define AE models.

[12]: LeDell et al. (2018), *h2o: R Interface for 'H2O'*

[13]: Dubossarsky et al. (2015), *autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data*

II.4 Illustrative Examples

An easy way to start using `ruta` is by means of the `autoencode` function. This will take a dataset and automatically train a simple AE and produce a codification for it. The function accepts several parameters, from which only the desired dimension is mandatory. Other optional parameters are the type of AE, the activation function in the middle layer and the number of epochs for the training process. The following example uses this function to extract 2 features from the well-known toy dataset Iris:

```r
library(ruta)
library(purrr)

encoded <- iris[, 1:4] %>% as.matrix() %>% autoencode(2, "robust")
```

These 2 features can be visualized like in Fig. II.2 in order to represent the model learned by the AE.

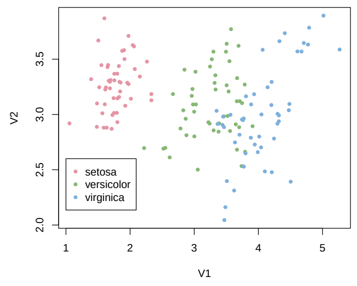


Figure II.2: Features learned by a basic AE with Iris data.

* https://ruta.software

The next step in difficulty involves defining a deep autoencoder. To help beginners describe its architecture, `ruta` provides a conversion from integer vector to neural network architecture in the following manner: `c(64, 16)` would become a network with an input layer the size of the inputs, a hidden layer with 64 variables, another hidden layer with 16 units for the encoding, the last hidden layer with 64 variables and an output layer the same size of the input one. Thus, the interface allows for simpler code, which can be observed in the following comparison between the code needed to define the same model in `ruta` and Keras:

```r
xtrain <- quakes[1:750,] %>% as.matrix()
xtest <- quakes[751:1000,] %>% as.matrix()
code_dim <- 2
hidden_dim <- 6

# ============== Ruta  ==============
features <- autoencoder(c(hidden_dim, code_dim), "sigmoid") %>%
  train(xtrain) %>%
  encode(xtest)

# ============== Keras ==============
input_l <- layer_input(shape = 5)
encoded <- layer_dense(input_l, units = hidden_dim)
encoded <- layer_dense(encoded, units = code_dim, activation = "sigmoid")
decoded <- layer_dense(encoded, units = hidden_dim)
decoded <- layer_dense(decoded, units = 5)

autoe <- keras_model(input_l, decoded)
encoder <- keras_model(input_l, encoded)
compile(autoe, loss = "mean_squared_error", optimizer = "rmsprop")
fit(autoe, xtrain, xtrain)
features <- predict(encoder, xtest)
```

The following example loads a dataset from Keras and normalizes its variables. Afterwards it defines a sparse AE by means of the `autoencoder_sparse` function with a 3-variable encoding, trains it and uses it to reconstruct test data. An evaluation is performed according to the mean squared error metric for the same test data.

```r
boston <- keras::dataset_boston_housing()

train_x <- scale(boston$train$x)
test_x <- scale(
  boston$test$x,
  center = train_x %@% "scaled:center",
  scale = train_x %@% "scaled:scale"
)

learner <- autoencoder_sparse(
  input() + dense(3, "tanh") + output(),
  "mean_squared_error"
)
model <- train(learner, train_x, epochs = 200)

reconstructions <- reconstruct(model, test_x)
evaluate_mean_squared_error(model, test_x)
```

Another task that can be performed by a trained variational AE is generation of new instances. In this case, we load the MNIST dataset of handwritten digits and learn 10 features which can be sampled via the `generate` function. Instances can also be generated by interpolating encodings from existing instances and decoding those interpolations, as Fig. II.3 shows.

```
mnist = keras::dataset_mnist()

x_train <- keras::array_reshape(
  mnist$train$x, c(dim(mnist$train$x)[1], 784)
) / 255.0
x_test <- keras::array_reshape(
  mnist$test$x, c(dim(mnist$test$x)[1], 784)
) / 255.0

network <-
  input() +
  dense(256, "elu") +
  variational_block(10, seed = 42) +
  dense(256, "elu") +
  output("sigmoid")
learner <- autoencoder_variational(network, loss = "binary_crossentropy")
model <- train(learner, x_train, epochs = 10)

samples <- model %>% generate(dimensions = c(8, 5), side = 6, fixed_values =
      0.99)
```

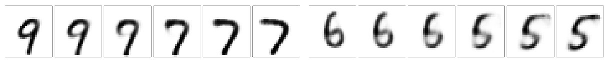


Figure II.3: Instances generated when interpolating between test samples in a variational AE trained with MNIST data.

The generic AE templates provided within the package may not always be adaptable enough for some problems. Thus, in order to provide detailed control over the model for more advanced users with some knowledge of Keras, `ruta` can convert its AE objects into a list of Keras models. This list contains three models: one for the encoder, another one for the decoder and one for the full AE. It can be accessed by setting the input shape in the Ruta object and calling the `to_keras` method:

```
obj <- autoencoder_contractive(c(128, 16))
obj$input_shape <- 1000
models <- to_keras(obj)
print(models$autoencoder)
```

Individual examples for each AE type are provided in the online documentation, as well as detailed instructions on how to build more customized neural architectures.

II.5 Conclusions

In this paper, we have presented a novel software piece focused in the construction of AEs, the `ruta` package for R. As opposed to most software developed on this topic, `ruta` implements several well-known AE variants and can handle different datasets. The software is implemented on top of Tensorflow and Keras in order to provide good performance, but abstracts many common aspects of AEs in order to provide an easy-to-use interface, accessible to R users with or without a programming background.

We have provided examples on how trained AEs can perform several tasks such as encoding and reconstruction of new data, as well as evaluation and even instance generation. When users need more control over the automatic generation of AE architectures, the package allows to extract the associated Keras models so as not to hinder their customization.

Since its publication on CRAN in May 2018 to the end of the year, `ruta` has received more than a thousand downloads from the RStudio CRAN mirror. Fig. II.4 shows the amount of downloads since the day of publication.

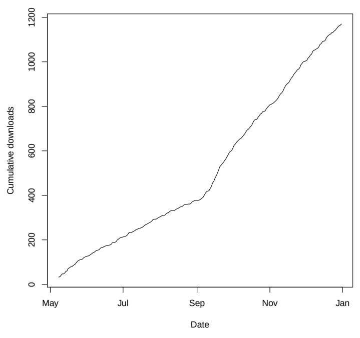


Figure II.4: Cumulative downloads since `ruta` was published.

Some supplementary software packages have already been planned. These include a package dedicated to visualizing the behavior of AEs, from their training process to the learned model, and a web-based user interface with the aim of providing easier access to these neural architectures.

Acknowledgements

This work was partially funded by the grant Iniciación a la Investigación para Alumnos de Máster of the University of Granada, project TIN2015-68854-R (FEDER Founds) of the Spanish Ministry of Economy and Competitiveness and project BigDaP-TOOLS - Ayudas Fundación BBVA a Equipos de Investigación Científica 2016.

References

[1] David Charte, Francisco Herrera, and Francisco Charte. “Ruta: Implementations of neural autoencoders in R”. In: *Knowledge-Based Systems* 174 (2019), pp. 4–8. DOI: 10.1016/j.knosys.2019.01.014.

[2] Ian T Jolliffe. *Principal component analysis*. Springer, 1986.

[3] Warren S Torgerson. “Multidimensional scaling: I. Theory and method”. In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916.

[4] Joshua B Tenenbaum, Vin de Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.

[5] Sam T Roweis and Lawrence K Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[6] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Deep Learning”. In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. Deep generative models, pp. 651–716.

[8] G. E. Hinton. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (July 28, 2006), pp. 504–507. DOI: 10.1126/science.1127647. (Visited on 08/26/2017).

[9] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[10] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, et al. “TensorFlow: A System for Large-scale Machine Learning”. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI’16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[11] David Charte, Francisco Charte, Salvador García, Mará J. del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007.

[12] Erin LeDell et al. *h2o: R Interface for 'H2O'*. R package version 3.20.0.2. 2018.

[13] Eugene Dubossarsky and Yuriy Tyshetskiy. *autoencoder: Sparse Autoencoder for Automatic Learning of Representative Features from Unlabeled Data*. R package version 1.1. 2015.

A snapshot on nonstandard supervised learning problems III

Source

David Charte, Francisco Charte, Salvador García, and Francisco Herrera. "A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations". In: *Progress in Artificial Intelligence* 8.1 (2019), pp. 1–14. DOI: 10.1007/s13748-018-00167-7

Abstract

Machine learning is a field which studies how machines can alter and adapt their behavior, improving their actions according to the information they are given. This field is subdivided into multiple areas, among which the best known are supervised learning (e.g. classification and regression) and unsupervised learning (e.g. clustering and association rules).

Within supervised learning, most studies and research are focused on well known standard tasks, such as binary classification, multiclass classification and regression with one dependent variable. However, there are many other less known problems. These are what we generically call nonstandard supervised learning problems. The literature about them is much more sparse, and each study is directed to a specific task. Therefore, the definitions, relations and applications of this kind of learners are hard to find.

The goal of this paper is to provide the reader with a broad view on the distinct variations of nonstandard supervised problems. A comprehensive taxonomy summarizing their traits is proposed. A review of the common approaches followed to accomplish them and their main applications is provided as well.

Keywords

Machine learning - Supervised learning - Nonstandard learning

III.1 Introduction

[2]: Mitchell (1997), *Machine learning*

According to Mitchell [2], a machine is said to learn from experience E related to a class of tasks T and performance metric P, when its performance at tasks in T improves according to P after experience E.

[3]: Marsland (2014), *Machine Learning: An Algorithmic Perspective*

Supervised learning is one of the fundamental areas of machine learning [3]. From object detection to ecological modeling to emotion recognition, it covers all kinds of applications. It essentially consists in learning a function by training with a set of input-output pairs. The training stage can be seen as E in the previous definition, and the specific task T may vary, but usually involves predicting an appropriate output given a new input.

[4]: Fukunaga (2013), *Introduction to statistical pattern recognition*
[5]: Jain et al. (2000), "Statistical pattern recognition: A review"

Traditionally, supervised learning problems have been spread into two categories: classification and regression [4, 5]. In the first, information is divided into discrete categories, while the latter involves patterns associated to a value in a continuous spectrum.

These problems can be processed by learning from a training dataset, which is composed of instances. Typically, these instances or samples take the form (x, y) where x is a vector of values in the space of input variables and y is a value in the target variable. Each problem can be described by the type of its instances: inputs will usually belong to a subset of $\mathbb{R}^n$, and outputs will take values in a specific one-dimensional set, finite or continuous. Once trained, the obtained model can be used to predict the target variable on unseen instances.

[6]: Duda et al. (2012), *Pattern classification*
[7]: Tax et al. (2002), "Using two-class classifiers for multiclass classification"

Standard classification problems are those where labels are either binary or multiclass [6, 7]. In the binary case, an instance can only be associated with one of two values: positive or negative, which is equivalent to 0 or 1. For example, email messages may be classified into spam or legit, and tumours can be categorized as either benign or malign. Multiclass problems, on the other hand, involve any finite number of classes. That is, any given instance will belong to one of possibly many categories, which is equivalent to it being assigned a natural number below a convenient threshold. As an example, a photograph of a plant or a sound recording from an animal could correspond to one of a variety of species.

[8]: James et al. (2013), *An Introduction to Statistical Learning: with Applications in R*
[9]: Smola et al. (1998), "On a kernel-based method for pattern recognition, regression, approximation, and operator inversion"

A standard regression problem [8, 9] consists in finding a function which is able to predict, for a given example, a real value among a continuous range, usually an interval or the set of real numbers $\mathbb{R}$.

For example, the height of a person may be estimated out of several characteristics such as age or country of origin.

Even though these standard problems are applicable in a multitude of cases, there are situations whose correct modeling requires modifications of their structure. For example, a newspaper article can be categorized according to its contents, but it could be desirable to assign several categories simultaneously. Similarly, a social media post could be described by not one but two input vectors, an image and a piece of text. These special circumstances cannot be covered by the traditional one-vector input and one-dimensional output schema. As a consequence, since performance metrics which measure improvements in standard tasks assume the common structure, they lose applicability or sense in these cases. Thus, not only new techniques are needed to tackle the problems, but also new ways of measuring and comparing their success.

This work studies variations on classic supervised problems where the traditional structure is not obeyed, which we call nonstandard variations. These emerge when the structure of the classical components of the problems does not suffice to describe complex situations, such as multiplicity of inputs or outputs, or order restrictions. As a consequence, this manuscript does not cover other singular supervised problems, such as high dimensionality of the feature space [10] or unbalanced training sets [11, 12], nor time-dependent problems, such as data streams [13, 14] or time series [15].

[10]: Bolón-Canedo et al. (2015), *Feature Selection for High-Dimensional Data*

[11]: Fernández et al. (2018), *Learning from Imbalanced Data Sets*

[12]: Krawczyk (2016), "Learning from imbalanced data: open challenges and future directions"

[13]: Gama (2010), *Knowledge discovery from data streams*

[14]: Silva et al. (2013), "Data stream clustering: A survey"

[15]: Hyndman et al. (2018), *Forecasting: principles and practice*

The rest of the paper is structured as follows. Section III.2 formally defines and describes each nonstandard variation. This is followed by Section III.3 establishing relations among the introduced problems and proposing a taxonomy of them. Section III.4 describes the most common techniques used to solve them. After that, Section III.5 enumerates popular applications of each problem. Section III.6 covers other variations further from the ones previously detailed. Lastly, Section III.7 draws some conclusions.

III.2 Definitions of nonstandard variations

The problems introduced in this section are generalizations over the traditional versions of classification and regression. The focus is on fully supervised problems, where inputs are always paired with outputs during training. An alternative taxonomy based on different supervision models is introduced in [16].

[16]: Hernández-González et al. (2016), "Weak supervision and other non-standard classification problems: A taxonomy"

Notation

In this work we will establish a notation which intends to be as simple to understand as possible, while being able to encompass every nonstandard variation. First, any supervised learning problem consists in finding a function which will classify, rank or perform regression. It will be noted as

$$f : X \to Y \tag{III.1}$$

where X is an input set, or domain, and Y is an output set, or codomain. It will be assumed that a training dataset S is provided, including a finite number of input-output pairs:

$$(x, y) \in S \subset X \times Y \; . \tag{III.2}$$

This way, a learning algorithm will be able to generate the desired function f. An additional notation will be the set of labels $\mathscr{L}$ where convenient.

For example, in standard binary classification $X \subset \mathbb{R}^n$ and $Y = \mathscr{L} = \{0, 1\}$. Similarly, standard regression problems can be defined with the same kind of X set and $Y \subset \mathbb{R}$. Thus, we can define very distinct supervised problems by particularizing sets X or Y in different ways.

[17]: Williams et al. (1998), "Bayesian classification with Gaussian processes"
[18]: Murphy (2012), *Machine Learning: A Probabilistic Perspective*

Other usual notations are based in probability theory, thus involving random variables and probability distributions [17, 18]. In that case, X and Y would be the sample spaces of the input and output variables $\mathbf{X}$ and $\mathbf{Y}$, respectively. Predictors would usually attempt to infer a discriminant model $P(\mathbf{Y}|\mathbf{X})$ from the training dataset.

Multi-instance

[19]: Herrera et al. (2016), *Multiple instance learning: foundations and algorithms*

The multi-instance (MI) framework [19] assumes a single feature space for all instances, but each training pattern may consist of more than one instance. In this case, a training pattern is composed of a finite multiset or *bag* of instances and a label. Formally, assuming instances are drawn from a set $A \subset \mathbb{R}^n$, the domain can be described as follows:

$$X = \{b \subset A \mid b \text{ finite}\} \; . \tag{III.3}$$

In this case, the learning algorithm will not know labels associated to each instance but to a bag of them. In addition to this, not all instances may share the same relevance or are equally related to the label.

Some MI problems assume that hidden labels are present for each instance in a bag: for example, a training set of drug tests where, for each test, several drug types are analyzed. Additionally, a typical MI assumption in the binary scenario states that a bag is positive when at least one of its instances is positive, and it is negative otherwise [20].

[20]: Foulds et al. (2010), "A review of multi-instance learning assumptions"

Other MI problems differ in that a per-instance labeling may not be possible or may not make sense: for example, if each bag represents an image and instances are image segments, class *beach* can only apply to bags with water and sand segments, but it cannot apply to an individual instance.

Multi-view

A learning problem is considered to be multi-view (MV) [21] when inputs are composed of several components of very different nature.

[21]: Zhao et al. (2017), "Multi-view learning overview: Recent progress and new challenges"

For example, if a learning pattern consists of an image as well as a piece of text representing the same instance, they can be seen as two *views* on it. In that case, images and texts would belong to distinct feature spaces A and B respectively, an input pattern being $(a, b) \in A \times B$. More generally, we can describe the input space as:

$$X = \prod_{i=1}^{t} A_i \text{ , where } A_i \subset \mathbb{R}^{n_i}, \tag{III.4}$$

where t is the number of views offered by the problem and n_i is the dimension of the feature space of the i-th view.

Multi-label

The multi-label (ML) learning field [22, 23] studies problems related to simultaneously assigning multiple labels to a single instance. That is, if $\mathcal{L} = \{l_1, \ldots, l_p\}$ the codomain consists of all possible selections of these p labels, also known as *labelsets*:

[22]: Herrera et al. (2016), *Multilabel classification*
[23]: Gibaja et al. (2015), "A tutorial on multilabel learning"

$$Y = 2^{\mathcal{L}} \cong \{0, 1\}^p \ . \tag{III.5}$$

As shown by this formulation, it is equivalent to think of a selection of labels as a subset of $\mathcal{L}$ and as a binary vector. For example, the labelset composed of the first and third labels can be represented either by $\{l_1, l_3\}$ or $(1, 0, 1, 0, \dots, 0)$.

The difference that arises when comparing ML problems to binary or multiclass ones is that labels may interact with each other. For example, a news piece classified in *economy* is more likely to be labeled *politics* than *sports*. Similarly, a photograph labeled *ocean* is less likely to have the *mountains* label rather than *beach*. Methods may take advantage of label co-ocurrence [24] in order to reduce the search space when predicting a labelset.

[24]: Charte et al. (2017), "Dealing with difficult minority labels in imbalanced mutilabel data sets"

A constrained version of ML classification is hierarchical ML classification [25], where labels are organized in a class hierarchy, usually a tree or a direct acyclic graph. A predicted labelset for a given instance is only consistent if parents of all labels in the labelset are also predicted.

[25]: Silla et al. (2011), "A survey of hierarchical classification across different application domains"

Multi-dimensional

Multi-dimensional (MD) learning [26] is a generalized classification problem where categorization is performed simultaneously along several dimensions. Each instance can belong to one of many classes in each dimension, thus the output space can be formally described as:

$$Y = \mathscr{L}_1 \times \mathscr{L}_2 \times \cdots \times \mathscr{L}_p, \tag{III.6}$$

where $\mathscr{L}_i$ is the label space for the i-th dimension.

[26]: Shatkay et al. (2008), "Multi-dimensional classification of biomedical text: Toward automated, practical provision of high-utility text to diverse users"

As with ML learning, label dimensions may be related in some way and treating them independently would only be a naive solution to the problem.

Label distribution learning

In label distribution learning (LDL) problems [27], otherwise known as probabilistic class label problems [28], any instance can be described in different degrees by each label. This can be modeled as a discrete distribution over the labels, where the probability of a label given a specific instance is called its *degree of description*. Analitically, the objective is, for each instance, to predict a real-valued vector which sums exactly 1:

[27]: Geng (2016), "Label distribution learning"

[28]: López-Cruz et al. (2013), "Learning conditional linear Gaussian classifiers with probabilistic class labels"

$$Y = \left\{ y \in [0,1]^p : \sum_{i=1}^{p} y_i = 1 \right\} . \tag{III.7}$$

In this case, we would say that the i-th label in $\mathscr{L}$ describes an instance (x, y) with degree y_i.

Label ranking

In a label ranking (LR) problem [29, 30] the objective is not to find a function able to choose one or several labels from the label space. Instead, it must evaluate their relevance for each unseen instance. The most general version of the problem involves a training set where Y is the set of all partial orders of $\mathscr{L}$, and the obtained function also maps individual instances to partial orders. This way, for each test instance the function will output a sequence of preferences where some labels will be seen as more relevant than others.

[29]: Hüllermeier et al. (2008), "Label ranking by learning pairwise preferences"
[30]: Vembu et al. (2010), "Label ranking algorithms: A survey"

However, the typical situation in label ranking problems is that the orders are total, which means any two labels can always be compared. This is called a *ranking* and does not exclude the possibility of ties. When ties are not allowed it is said to be a *sorting* or *permutation*, and can be formulated as follows:

$$Y = \{\sigma : \{1, \ldots, p\} \to \mathscr{L} \mid \sigma \text{ is bijective}\} \ , \tag{III.8}$$

where p is the amount of labels. Y can also be seen as the set of all permutations of the labels in $\mathscr{L}$, usually known as the symmetric group of order p, and noted as S_p.

Multi-target regression

A regression problem where the output space has more than just one dimension is usually called multi-target regression (MTR) and is also known as multi-output, multi-variate or multi-response [31]. In this case, a formal description is simply that the codomain is a continuous multi-dimensional real set:

[31]: Borchani et al. (2015), "A survey on multi-output regression"

$$Y = \prod_{i=1}^{p} Y_i \ , \text{ where } Y_i \subset \mathbb{R} \ \forall i \tag{III.9}$$

and p is the number of target variables.

As with other multiple target extensions, the key difference with single-target regression in this case is the possible interactions among output variables.

Ordinal regression

A problem where the target space is discrete but ordered is called ordinal regression (OR) or, alternatively, ordinal classification [32]. It can be located midway between classification and regression. More specifically, it consists in labeling instances with a finite number of choices where these are ordered

[32]: Gutiérrez et al. (2016), "Ordinal Regression Methods: Survey and Experimental Study"

$$Y = \{1, 2, \ldots, c\}\ ,\ 1 < 2 < \cdots < c\ . \tag{III.10}$$

In OR, the training phase consists in learning from a set of feature vectors which have a specific label associated to them, and testing can be performed over individual instances. This means that, although labels are ordered, the main objective is not to rank or sort instances as in learning to rank [33], but to simply classify them. The labels themselves do not provide any metric information either, they only carry qualitative information about the order among themselves.

[33]: Burges et al. (2005), "Learning to rank using gradient descent"

Monotonicity constraints

Order relations can exist not only in the label space but in the feature space as well. Partial orders among real-valued feature vectors are always possible, and there may be cases where the order among instances is determined by just one or a few of their attributes.

When inputs as well as outputs are at least partially ordered, it is common to look for predictions which respect their order relations. In that case, the objective is to obtain a classifier or regression function which enforces the following constraint:

$$x_1 < x_2 \Rightarrow f(x_1) < f(x_2)\ \forall x_1, x_2 \in X\ . \tag{III.11}$$

When Y is discrete the problem is usually called *monotone classification* (MC), monotonic classification or ordinal classification with monotonicity constraints [34]. If, on the contrary, Y is continuous, it is known as *isotonic regression* (IR) [35].

[34]: Gutiérrez et al. (2016), "Current prospects on ordinal and monotonic classification"

[35]: Barlow (1972), *Statistical inference under order restrictions; the theory and application of isotonic regression*

Absence or partiality of information

Some problems do not directly alter the structure of X and Y from the standard supervised problem. Instead, they restrict which data can belong to a training set, or remove labelings from training examples.

In this case, training information is presented partially or with some exclusions.

According to which kind of information is missing from the training set, a learning task can usually be categorized as semi-supervised [36], one-class learning [37], PU-learning [38], zero-shot learning [39] or one-shot learning [40]. These are described further in Section III.6.

[36]: Chapelle et al. (2010), *Semi-Supervised Learning*

[37]: Moya et al. (1993), "One-class classifier networks for target recognition applications"

[38]: Elkan et al. (2008), "Learning classifiers from only positive and unlabeled data"

[39]: Palatucci et al. (2009), "Zero-shot learning with semantic output codes"

[40]: Fe-Fei et al. (2003), "A Bayesian approach to unsupervised one-shot learning of object categories"

Variation combinations

Some of the components described above can be combined to compose a more complex problem overall. Usually, one of these combinations will take components from different variation types, for example, simultaneous multiplicity of inputs and outputs.

More specifically, there exist several studies involving MI ML scenarios [41, 42]. In this case, examples from the input space are composed of several feature vectors and are associated to various labels. As a consequence, this model can represent many complicated problems where inputs and outputs have more structure than usual.

[41]: Zhou et al. (2012), "Multi-instance multi-label learning"

[42]: Surdeanu et al. (2012), "Multi-instance multi-label learning for relation extraction"

Other more uncommon situations are MV MI ML problems [43], where patterns have several instances which may or may not belong to the same space, a multi-output version of OR named graded ML classification [44] and more complex input structures such as multi-layer MI MV [45], where a hierarchy of instances is present in each example.

[43]: Nguyen et al. (2014), "Labeling Complicated Objects: Multi-View Multi-Instance Multi-Label Learning."

[44]: Cheng et al. (2010), "Graded multilabel classification: The ordinal case"

[45]: Wu et al. (2014), "Music emotion recognition by multi-label multi-layer multi-instance multi-view learning"

III.3 Taxonomy

A first categorization of the variations analyzed in this work can be made according to how they differ from the standard problem. There can be multiplicity in the input space or the output space, order constraints may exist, or only partial information may be given in some cases. Fig. III.1 shows ways in which the traditional problems can be generalized.

Problems introducing multiple inputs are MI and MV, whereas multiple outputs can be found on ML, MD, LR, LDL and MTR. Problems where orders are present are OR, MC and IR. Likewise, tasks with only partial information are, among others, semi-supervised learning (SSL), positive-unlabeled (PU) learning, one-shot classification and zero-shot classification.

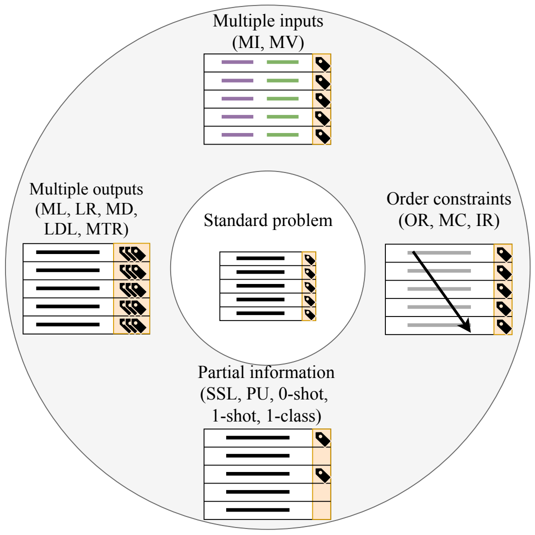


Figure III.1: Extensions of the standard supervised problem: multiple inputs or outputs, presence of orders and rankings, and partial information.

Finally, a generalized problem can be built out of combining several of these components: for example, a multiple-input multiple-output problem where the inputs and outputs can belong to structures like the ones defined above.

The rest of this section studies variations on the structure of the input space and output space, establishes relations among problems, and describes how they can be particularized or generalized to one another.

Input structure

In a standard supervised problem, the input space consists of single feature vectors and does not impose a specific order.

Problems where learning patterns are composed of multiple instances can usually be categorized into either MI, if the inputs share the same structure, or MV, otherwise. Their combination can also be considered as well, e.g. a problem where an example is composed of one or more photographs and one or more pieces of text. This would be a case of a MV MI problem.

There are also problems where there exists a partial or total order among instances, which is coupled with an order constraint in relation to the outputs. These are MC and IR.

Fig. III.2 summarizes these structural traits in a hierarchy and indicates problems where these traits are present.

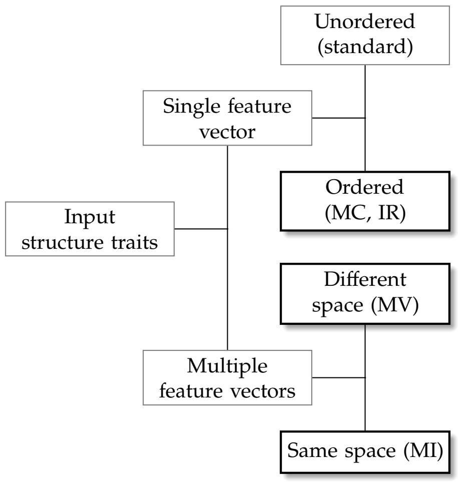


Figure III.2: Traits that can be found on the input structure of supervised problems.

Output structure

The diversity in output variations is higher than that of the input ones. A first sorting criterion is whether the codomain is discrete or continuous. This way, problems are either classification or regression ones.

Further subdivision of problems allows to separate these traits according to whether outputs remain scalars or become vectors. In the first case we consider order in the discrete scenario a nonstandard variation, which is present in OR and MC. In the second case, classification problems are spread into ML, LR and MD, and regression ones into LDL and MTR.

Fig. III.3 organizes these traits in a hierarchy based on the previous criteria. Each leaf of the tree also includes problems where each one is present.

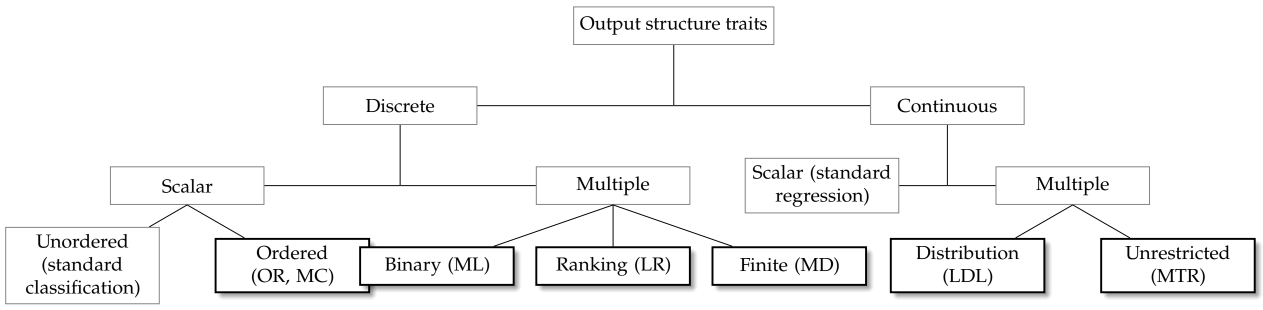


Figure III.3: Traits that can be found on the output structure of supervised problems.

The variations in the structure of target spaces in supervised problems can be seen as generalizations of the standard problems. Furthermore, some of them are also more general than others. For example, ML problems can be seen as LR ones where, for a given

Table III.1: Identification of problems according to their input traits (vertical axis) and output traits (horizontal axis).

Inputs \ Outputs	Unordered outputs		Ordered outputs			
	Scalar	Multiple	Scalar		Multiple	
			Discrete	Continuous	Discrete	Continuous
Unordered inputs	standard classification [4]	ML/MD classification [22, 26]	OR [32]	standard regression [9]	Graded ML [44]	MTR [31]
Ordered inputs	-	-	MC [34]	IR [35]	-	-
Multiple instances	MI classification [19]	MIML/MIMD classification [41]	-	MI regression [19]	-	-
Multiple views	MV classification [21]	MVML/MVMD classification [43]	-	MV regression [21]	-	-

instance, labels over a threshold are active and those below are not. Thus, LR is a generalization of the ML scenario. More relations of this kind are displayed in Fig. III.4.

As shown in the graph, an inclusion of more target variables of the same type transforms a binary problem into ML, a multiclass problem into MD and a single-target regression one into MTR. Similarly, inclusion of more values into each variable allows to generalize binary problems to multiclass, and ordinal to single-target regression, as well as ML ones to MD and these to MTR. LDL can be seen as a generalization of ML where real numbers between 0 and 1 are also allowed as values for a label. LR is a generalization of ML by the argument discussed before.

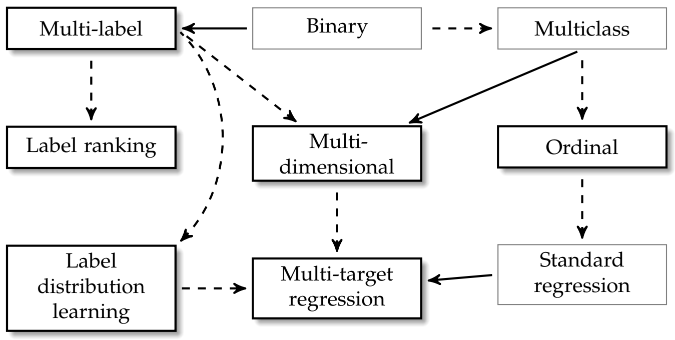


Figure III.4: Relations among supervised problems according to output structure. Arrows follow natural generalizations from one problem to another. Continuous arrows denote generalizations based on adding more variables of the same type. Dashed arrows indicate generalizations based on modifying existing target variables.

Summary

In this section input and output variations of standard supervised problems have been categorized and related. Table III.1 allows to identify specific problems according to which input and output traits are present.

III.4 Common approaches to tackle nonstandard problems

When tackling a nonstandard problem, most techniques follow one of two main approaches: problem transformation or algorithm adaptation. The first one relies on appropriate transformations of the data which result in one or more simpler, standard problems. The latter implies an extension or development of previously existing algorithms, in order to adapt them to the complexities induced by the structure of the data.

In the following subsections several methods based on both approaches are enumerated for each analysed problem.

Problem transformation

Problem transformation methods assume that a solution can be achieved by extracting one or more simpler problems out of the original one. For example, a problem with multi-dimensional targets could be transformed into many problems with scalar outputs. Then, these problems could be solved independently by a classical algorithm. A solution for the original problem would be the concatenation of those extracted from the simpler ones.

Next, the most common transformation techniques are described for each nonstandard supervised learning task previously introduced.

– MI. The taxonomy proposed in [46] describes an Embedded Space paradigm, where each bag is transformed into a single feature vector representing the relevant information about the whole bag. This transformation brings the MI problem into a single-instance one. Most of these methods are vocabulary-based, which means that the embedding uses a set of concepts to classify each bag according to its instances, resulting in a single vector with one component per concept.

– MV. Some naive transformations consist in ignoring every view except one, or concatenating feature vectors from all views, thus training a single-view model in both cases [47]. A preprocessing based on Canonical Correlation Analysis [48] is able to project data from multiple views onto a lower-dimensional, single-view space.

– ML. Transformation methods for ML classification [49] are diverse: Binary Relevance trains separate binary classifiers for each label. Label Powerset reduces the problem to a multiclass one by treating each individual labelset as an independent class label, and Random k-Labelsets [50] extracts an ensemble of multiclass problems similarly. Classifier chains [51] trains subsequent binary classifiers accumulating previous predictions as inputs. ML problems can also be transformed to LR [52].

– MD. In some cases, independent classifiers can be trained for several dimensions [26, 53] but this method ignores possible correlations among dimensions. An alternative transformation, building a different label from each combination of classes, would produce a much larger label space and thus is not typically applied.

– LDL. A LDL problem can be reduced to multiclass classification by extracting as many single-label examples as labels for each one of the training instances [27]. These new examples are assigned a class corresponding to each label and weighted according to its degree of description. During the prediction process, the classifier must be able to output the score/confidence for each label, which can be used as its description degree.

– LR. A reduction of this problem to several binary problems can be achieved by learning pairwise preferences [29]. This transforms a c-label problem into $c(c-1)/2$ binary problems describing a comparison among two labels. An alternative reduction by means of constraint classification [54] builds a single binary classification dataset by expanding each label preference into a new positive instance and a new negative instance. The feature space of the new binary problem has dimension nc, where n is the original dimension and c the number of labels, due to the constraints embedded in it by Kesler's construction [55].

– MTR. There are several ways to transform a MTR problem into several single-target regression ones. Some of them are inspired by the ML field, such as a one-vs-all single-target reduction, multi-target stacking and regressor chains [56]. All of them train single-target regressors for several extracted problems, and then combine the obtained predictions. A different approach based on support vectors [57] extends the feature space which expresses the multi-output problem as a single-target one that can be solved using least squares support vector regression machines.

– OR. An ordinal problem with c classes can be transformed into $c - 1$ binary classification problems by using each class from the second to the last one as a threshold for the positive class [58]. This decomposition can be called *ordered partitions* and is not the only possible one: others are *one-vs-next*, *one-vs-followers* and *one-vs-previous* [32]. Several 3-class problems can also be obtained by using, for the i-th problem, classes "l_i", "$< l_i$" and "$> l_i$".

– MC. The authors in [59] describe a procedure to tackle binary MC problems by means of IR. Multiclass MC cases can be reduced to several binary MC ones, which in turn are solved as IR problems.

Algorithm adaptation

Existing methods for classical problems can be extended in order to introduce the necessary complexities of nonstandard variations. As an example, nearest neighbor methods could be coupled with new distance metrics in order to be able to measure similarity among multiple inputs.

The rest of this section presents some algorithm adaptations which can be used to tackle nonstandard supervised tasks.

– MI. Methods that work on instance level are adaptations of algorithms from single-instance classification whose responses are then aggregated to build the bag-level classification [46]. They typically assume that one positive instance implies a positive bag. Adaptations of common algorithms have been proposed with support vector machines (SVM) [60] and neural networks [61], whereas some original methods in this area are Axis-Parallel Rectangles [62] and Diverse Density [63]. In the bag-space paradigm, methods treat bags as a whole and use specific distance metrics with distance as well as kernel-based classifiers, such as k-nearest neighbor (k-NN) [64] or SVM [65].

– MV. Supervised methods for MV are comparatively less developed than semi-supervised ones. Nonetheless, there is an extension of SVM [66] which simultaneously looks for two SVMs, one in each of the feature spaces of a two-view problem. There is an extension of Fisher discriminant analysis as well [67].

– ML. The most relevant algorithm adaptations [49] are based on standard classification algorithms with added support for choosing more than one class at a time: adaptations exist for k-NN [68], decision trees [69], SVMs [70], association rules [71] and ensembles [72].

– MD. Specific Bayesian networks have been proposed for the MD scenario [73, 74], as well as Maximum Entropy-based algorithms [26, 53].

– LDL. Proposals in [27] are adaptations of k-NN, with a special derivation of the label distribution of an unseen instance given its neighbors, and backpropagated neural networks, where the output layer indicates the label distribution of an instance. Other proposed methods are based on the optimization algorithms BFGS and Improved Iterative Scaling.

– LR. Boosting methods have been adapted to LR [75], as well as the SVM proposed in [70] for ML which can be naturally extended to LR [30]. An adaptation of online learning algorithms such as the perceptron has also been developed [76].

– MTR. First methods able to treat MTR problems were actually generalizations of statistical methods for single-target regression [77, 78]. Other common methods which have been extended to predict multiple regression variables are support vector regression [79, 80], kernel-based methods [81, 82], and regression trees [83] as well as random forests [84].

– OR. Neural networks can be used to tackle OR with slight changes in the loss function or the output layer [85, 86]. Similarly, extreme learning machines have also been applied to this problem [87, 88]. Common techniques such as k-NN or decision trees have been coupled with global constraints for OR [89], and extensions of other well known algorithms such as Gaussian processes [90] and AdaBoost [91] have been proposed as well.

– MC. Algorithm adaptations generally take a well known technique and add monotonicity constraints. For example, there exist in the literature adaptations of k-NN [92], decision trees [93], decision rules [94, 95] and artificial neural networks [96].

Table III.2 gathers all the methods described previously to tackle nonstandard supervised tasks.

Table III.2: Summary table of presented methods according to their type of approach.

Task	Problem transformation	Algorithm adaptation
MI	Embedded-space [46]	SVM [60, 65] Neural networks [61] k-NN [64]
MV	Canonical correlation analysis [48]	SVM [66] Fisher discriminant analysis [67]
ML	Binary Relevance [49] Label Powerset [49] Classifier chains [51]	k-NN [68] Decision trees [69] SVM [70] Association rules [71] Ensembles [72]
MD	Independent classifiers [26, 53]	Bayesian networks [73, 74] Maximum Entropy [26, 53]
LDL	Multiclass reduction [27]	k-NN [27] Neural networks [27]
LR	Pairwise preferences [29] Constraint classification [54]	Boosting [75] SVM [30] Perceptron [76]
MTR	ML inspired: one-vs-all, stacking, regressor chains [56] Support vectors [57]	Generalizations [77, 78] Support vector regression [79, 80] Kernel-based [81, 82] Regression trees [83] Random forests [84]
OR	Ordered partitions [58] One-vs-next, One-vs-followers, One-vs-previous [32] 3-class problems [32]	Neural networks [85, 86] Extreme learning machines [87, 88] Decision trees [89] Gaussian processes [90] AdaBoost [91]
MC	Reduction to IR [59]	k-NN [92] Decision trees [93] Decision rules [94, 95] Neural networks [96]

III.5 Applications. Original real word scenarios

The problems studied in this work have their origins in real-world scenarios which are related below:

– **MI.** Problems modeled under MI learning are drug activity prediction [62], where each pattern describes a molecule and its different forms are represented by instances; image classification [46], and bankruptcy [97]. Most of the datasets used in experimentations, however, are usually synthetic.

– **MV.** Some situations where data is described in multiple views are multilingual text categorization [98], face detection with several poses [99], user localization in a WiFi network [100], advertisements described by their image and surrounding text [101] and image classification with several color-based views and texture-based views [102].

– **ML.** Problems which fall naturally under the ML definition are text classification under several categories simultaneously [103], image labeling [104], question tagging in forums where tags can co-exist [105], protein classification [106], data streams [107] and recommendation systems [108].

– **MD.** Applications of MD classification include classification of biomedical text [26], where predicted dimensions for a given document are its focus, evidence type, certainty level, polarity and trend; gene function identification [73]; tumor classification, and illness diagnosis in animals [74].

– **LR.** The field known as *preference learning* has been gaining interest [29], and LR is one of the problem that falls under this term. LR is also frequently applied in ML scenarios [109], where a threshold can be applied in order to transform an obtained ranking into a labelset.

– **LDL.** Data with relative importance of each label appears in applications such as analysis of gene expression levels in yeast [110], or emotion description from facial expressions [111], where a face can depict several emotions in different grades.

– **MTR.** Applications modeled as MTR problems are diverse, including modeling of vegetation condition in ecosystems assigning several scores which depend on the vegetation type [112], prediction of audio spectrums of wind tunnel tests [113], and estimation of several biophysical parameters from remote sensing images [114].

– OR. The most salient fields where OR can be found are text classification [115], where the predicted variable may be an opinion scale or a degree of satisfaction; image categorization [116]; medical research [117]; credit rating [118], and age estimation [119].

– MC. Monotonicity constraints are found in problems related to customer satisfaction analysis [120], in which overall appreciation of a product must increase along with the evaluation of its features; house pricing [93]; bankruptcy risk evaluation [121], and cancer prediction [122], among others.

III.6 Other nonstandard variations

This section covers variations of the standard supervised problem which are further from the central focus of this paper less related to those above.

Learning with partial information

In a standard supervised classification setting, it is assumed that every training example is labeled accordingly and that there exist examples for every class that may appear in the testing phase. When only a fraction of the training instances are labeled, the problem is considered semi-supervised [36], but generally there still exist labeled samples for each class.

[36]: Chapelle et al. (2010), *Semi-Supervised Learning*

In positive-unlabeled learning [38, 123], however, labeled examples provided within the training set are only positive. This means the learning algorithm only knows about the class of positive instances, and unlabeled ones can have either class.

[38]: Elkan et al. (2008), "Learning classifiers from only positive and unlabeled data"
[123]: Liu et al. (2003), "Building text classifiers using positive and unlabeled examples"

A different scenario arises when the training set only consists of negative (or only positive) instances, and no unlabeled examples are provided. This is known as one-class classification [37], and data of this nature can be obtained from outlier detection applications, where positive examples are hardly recorded.

[37]: Moya et al. (1993), "One-class classifier networks for target recognition applications"

A problem which may be seen as a generalization of one-class classification is zero-shot learning [39], a situation where unseen classes are to be predicted in the testing stage. That is, the label space $\mathcal{Y}$ includes some values which are not present in any training pattern, but the classifier must be able to predict them. For example, if in a speech recognition problem $\mathcal{Y}$ is the set of all words in English,

[39]: Palatucci et al. (2009), "Zero-shot learning with semantic output codes"

the training set is unlikely to have at least one instance for each word, thus the classifier will only succeed if it is capable of assigning unlearned words to test examples.

A relaxation on the obstacles of zero-shot learning is present in one-shot learning [40], where algorithms attempt to generalize from very few (1 to 5) examples of each class. This is a common circumstance in the field of image classification, where the cost of collecting and labeling data samples is high.

[40]: Fe-Fei et al. (2003), "A Bayesian approach to unsupervised one-shot learning of object categories"

A classification of these problems according to the type of missing information can be found in Table III.3.

Table III.3: Partial information problems according to the kind of absence in the training set.

Trait	**Problem types**
Presence of unlabeled instances	Semi-supervised [36], Positive-unlabeled [38]
No representation of some classes	One-class [37], Positive-unlabeled [38], Zero-shot [39]
Scarce representation of some classes	One-shot [40]

Prediction of structured data

The nonstandard variations described in this work generalize traditional supervised problems where the predicted output is at most a vector whose components take values in either a finite set or $\mathbb{R}$. Further generalizations are possible if other kinds of structures are allowed. For example, the target may take the form of an ordered sequence or a tree. In this case, the problem usually enters the scope of structured prediction [124], a generalization of supervised learning where methods must build structured data associated to input instances.

[124]: Taskar et al. (2005), "Learning structured prediction models: A large margin approach"

A particular case of supervised problem which can be seen under the umbrella of structured prediction is learning to rank [33], which does not involve a label space as such. Instead, training consists in learning from a set of feature vectors with a series of preferences among them, that is, a partial or total order in the training set. During testing a set of feature vectors is provided and the desired output is a ranking (with a predefined number of relevance levels, allowing ties) or a sorting (simply an ordering of the instances). This problem differs from OR in that individual classifications are usually meaningless: only relative distances among ranked instances matter.

[33]: Burges et al. (2005), "Learning to rank using gradient descent"

III.7 Conclusions

Traditional supervised learning comprises two well known problems in machine learning: classification and regression. However, the multitude of applications which do not strictly fit the structure of the standard versions of those problems have favored the development of alternative versions which are more flexible and allow the analysis of more complex situations.

In this work an overview of nonstandard variations of supervised learning problems has been presented. A novel taxonomy under several criteria has described relationships among these variations, where the main differentiating properties are multiplicity of inputs, multiplicity of outputs, presence of order relations and constraints, and partial information. Afterwards, common methods for tackling these problems have been outlined and their main applications have been mentioned as well. Finally, some additional variants which were left out of the scope of the previous analysis have been introduced as well.

Design of novel algorithms for nonstandard supervised tasks is scarcer than adaptations and transformations, but there exist some approximations and even more open possibilities for tackling these from classical algorithmic perspectives, such as probabilistic and heuristic methods, information theory and linear algebra, among others.

Acknowledgments

D. Charte is supported by the Spanish Ministry of Science, Innovation and Universities under the FPU National Program (Ref. FPU17/04069). This work has been partially supported by projects TIN2017-89517-P (FEDER Founds) of the Spanish Ministry of Economy and Competitiveness and TIN2015-68454-R of the Spanish Ministry of Science, Innovation and Universities.

References

[1] David Charte, Francisco Charte, Salvador García, and Francisco Herrera. “A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations”. In: *Progress in Artificial Intelligence* 8.1 (2019), pp. 1–14. DOI: 10.1007/s13748-018-00167-7.

[2] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[3] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall, 2014.

[4] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013.

[5] Anil K Jain, Robert PW Duin, and Jianchang Mao. “Statistical pattern recognition: A review”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.1 (2000), pp. 4–37.

[6] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[7] David MJ Tax and Robert PW Duin. “Using two-class classifiers for multiclass classification”. In: *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 2. IEEE. 2002, pp. 124–127.

[8] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer New York, 2013.

[9] Alex J Smola and Bernhard Schölkopf. “On a kernel-based method for pattern recognition, regression, approximation, and operator inversion”. In: *Algorithmica* 22.1-2 (1998), pp. 211–231.

[10] Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. *Feature Selection for High-Dimensional Data*. Cham: Springer International Publishing, 2015.

[11] Alberto Fernández et al. *Learning from Imbalanced Data Sets*. Springer International Publishing, 2018.

[12] Bartosz Krawczyk. “Learning from imbalanced data: open challenges and future directions”. In: *Progress in Artificial Intelligence* 5.4 (Nov. 2016), pp. 221–232. DOI: 10.1007/s13748-016-0094-0.

[13] Joao Gama. *Knowledge discovery from data streams*. Chapman and Hall/CRC, 2010.

[14] Jonathan A Silva et al. “Data stream clustering: A survey”. In: *ACM Computing Surveys (CSUR)* 46.1 (2013), p. 13.

[15] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[16] Jerónimo Hernández-González, Iñaki Inza, and Jose A. Lozano. “Weak supervision and other non-standard classification problems: A taxonomy”. In: *Pattern Recognition Letters* 69 (2016), pp. 49–55. DOI: 10.1016/j.patrec.2015.10.008.

[17] Christopher KI Williams and David Barber. “Bayesian classification with Gaussian processes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.12 (1998), pp. 1342–1351.

[18] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[19] Francisco Herrera et al. *Multiple instance learning: foundations and algorithms*. Springer, 2016.

[20] James Foulds and Eibe Frank. “A review of multi-instance learning assumptions”. In: *The Knowledge Engineering Review* 25.1 (2010), pp. 1–25. DOI: 10.1017/S026988890999035X.

[21] Jing Zhao, Xijiong Xie, Xin Xu, and Shiliang Sun. “Multi-view learning overview: Recent progress and new challenges”. In: *Information Fusion* 38 (2017), pp. 43–54. DOI: 10.1016/j.inffus.2017.02.007.

[22] Francisco Herrera, Francisco Charte, Antonio J Rivera, and María J Del Jesus. *Multilabel classification*. Springer, 2016.

[23] Eva Gibaja and Sebastián Ventura. “A tutorial on multilabel learning”. In: *ACM Computing Surveys (CSUR)* 47.3 (2015), p. 52. DOI: 10.1145/2716262.

[24] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. “Dealing with difficult minority labels in imbalanced mutilabel data sets”. In: *Neurocomputing* (2017). DOI: 10.1016/j.neucom.2016.08.158.

[25] Carlos N Silla and Alex A Freitas. “A survey of hierarchical classification across different application domains”. In: *Data Mining and Knowledge Discovery* 22.1-2 (2011), pp. 31–72.

[26] Hagit Shatkay, Fengxia Pan, Andrey Rzhetsky, and W John Wilbur. “Multi-dimensional classification of biomedical text: Toward automated, practical provision of high-utility text to diverse users”. In: *Bioinformatics* 24.18 (2008), pp. 2086–2093. DOI: 10.1093/bioinformatics/btn381.

[27] Xin Geng. “Label distribution learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.7 (2016), pp. 1734–1748. DOI: 10.1109/TKDE.2016.2545658.

[28] Pedro L López-Cruz, Concha Bielza, and Pedro Larrañaga. “Learning conditional linear Gaussian classifiers with probabilistic class labels”. In: *Conference of the Spanish Association for Artificial Intelligence*. Springer. 2013, pp. 139–148. DOI: 10.1007/978-3-642-40643-0_15.

[29] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. “Label ranking by learning pairwise preferences”. In: *Artificial Intelligence* 172.16-17 (2008), pp. 1897–1916. DOI: 10.1016/j.artint.2008.08.002.

[30] Shankar Vembu and Thomas Gärtner. “Label ranking algorithms: A survey”. In: *Preference learning*. Springer, 2010, pp. 45–64. DOI: 10.1007/978-3-642-14125-6_3.

[31] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. “A survey on multi-output regression”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 216–233. DOI: 10.1002/widm.1157.

[32] P. A. Gutiérrez, M. Pérez-Ortiz, J. Sánchez-Monedero, F. Fernández-Navarro, and C. Hervás-Martínez. “Ordinal Regression Methods: Survey and Experimental Study”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.1 (2016), pp. 127–146. DOI: 10.1109/TKDE.2015.2457911.

[33] Chris Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 89–96. DOI: 10.1145/1102351.1102363.

[34] Pedro Antonio Gutiérrez and Salvador García. “Current prospects on ordinal and monotonic classification”. In: *Progress in Artificial Intelligence* 5.3 (Aug. 2016), pp. 171–179. DOI: 10.1007/s13748-016-0088-y.

[35] Richard E Barlow. *Statistical inference under order restrictions; the theory and application of isotonic regression*. Wiley, 1972.

[36] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010.

[37] Mary M Moya, Mark W Koch, and Larry D Hostetler. "One-class classifier networks for target recognition applications". In: *NASA STI/Recon Technical Report N* 93 (1993).

[38] Charles Elkan and Keith Noto. "Learning classifiers from only positive and unlabeled data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 213–220. DOI: 10.1145/1401890.1401920.

[39] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. "Zero-shot learning with semantic output codes". In: *Advances in neural information processing systems*. 2009, pp. 1410–1418.

[40] Li Fe-Fei et al. "A Bayesian approach to unsupervised one-shot learning of object categories". In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE. 2003, pp. 1134–1141. DOI: 10.1109/ICCV.2003.1238476.

[41] Zhi-Hua Zhou, Min-Ling Zhang, Sheng-Jun Huang, and Yu-Feng Li. "Multi-instance multi-label learning". In: *Artificial Intelligence* 176.1 (2012), pp. 2291–2320. DOI: 10.1016/j.artint.2011.10.002.

[42] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D Manning. "Multi-instance multi-label learning for relation extraction". In: *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics. 2012, pp. 455–465.

[43] Cam-Tu Nguyen, Xiaoliang Wang, Jing Liu, and Zhi-Hua Zhou. "Labeling Complicated Objects: Multi-View Multi-Instance Multi-Label Learning." In: *AAAI*. 2014, pp. 2013–2019.

[44] Weiwei Cheng, Eyke Hüllermeier, and Krzysztof J Dembczynski. "Graded multilabel classification: The ordinal case". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 223–230.

[45] Bin Wu, Erheng Zhong, Andrew Horner, and Qiang Yang. "Music emotion recognition by multi-label multi-layer multi-instance multi-view learning". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 117–126. DOI: 10.1145/2647868.2654904.

[46] Jaume Amores. "Multiple instance classification: Review, taxonomy and comparative study". In: *Artificial Intelligence* 201 (2013), pp. 81–105. DOI: https://doi.org/10.1016/j.artint.2013.06.003.

[47] Abhishek Kumar, Piyush Rai, and Hal Daume. "Co-regularized multi-view spectral clustering". In: *Advances in neural information processing systems*. 2011, pp. 1413–1421.

[48] Kamalika Chaudhuri, Sham M Kakade, Karen Livescu, and Karthik Sridharan. "Multi-view clustering via canonical correlation analysis". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 129–136. DOI: 10.1145/1553374.1553391.

[49] Min-Ling Zhang and Zhi-Hua Zhou. "A review on multi-label learning algorithms". In: *IEEE transactions on knowledge and data engineering* 26.8 (2014), pp. 1819–1837. DOI: 10.1109/TKDE.2013.39.

[50] Grigorios Tsoumakas and Ioannis Vlahavas. "Random k-labelsets: An ensemble method for multilabel classification". In: *European conference on machine learning*. Springer. 2007, pp. 406–417. DOI: 10.1007/978-3-540-74958-5_38.

[51] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. "Classifier chains for multi-label classification". In: *Machine learning* 85.3 (2011), p. 333. DOI: 10.1007/s10994-011-5256-5.

[52] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. "Multilabel classification via calibrated label ranking". In: *Machine learning* 73.2 (2008), pp. 133–153. DOI: 10.1007/s10994-008-5064-8.

[53] Fengxia Pan. *Multi-dimensional fragment classification in biomedical text*. Queen's University, 2006.

[54] Sariel Har-Peled, Dan Roth, and Dav Zimak. "Constraint classification for multiclass classification and ranking". In: *Advances in neural information processing systems*. 2003, pp. 809–816.

[55] Nils J Nilsson. *Learning machines: foundations of trainable pattern-classifying systems*. McGraw-Hill, 1965.

[56] Eleftherios Spyromitros-Xioufis, Grigorios Tsoumakas, William Groves, and Ioannis Vlahavas. "Multi-label classification methods for multi-target regression". In: *arXiv preprint arXiv* 1211 (2012).

[57] Wei Zhang, Xianhui Liu, Yi Ding, and Deming Shi. "Multi-output LS-SVR machine in extended feature space". In: *Computational Intelligence for Measurement Systems and Applications (CIMSA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 130–134. DOI: 10.1109/CIMSA.2012.6269600.

[58] Eibe Frank and Mark Hall. "A simple approach to ordinal classification". In: *European Conference on Machine Learning*. Springer. 2001, pp. 145–156. DOI: 10.1007/3-540-44795-4_13.

[59] W. Kotlowski and R. Slowinski. "On Nonparametric Ordinal Classification with Monotonicity Constraints". In: *IEEE Transactions on Knowledge and Data Engineering* 25.11 (2013), pp. 2576–2589. DOI: 10.1109/TKDE.2012.204.

[60] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. "Support vector machines for multiple-instance learning". In: *Advances in neural information processing systems*. 2003, pp. 577–584.

[61] Jan Ramon and Luc De Raedt. "Multi instance neural networks". In: *Proceedings of the ICML-2000 workshop on attribute-value and relational learning*. 2000, pp. 53–60.

[62] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. "Solving the multiple instance problem with axis-parallel rectangles". In: *Artificial intelligence* 89.1-2 (1997), pp. 31–71. DOI: 10.1016/S0004-3702(96)00034-3.

[63] Oded Maron and Tomás Lozano-Pérez. "A framework for multiple-instance learning". In: *Advances in neural information processing systems*. 1998, pp. 570–576.

[64] Jun Wang and Jean-Daniel Zucker. "Solving Multiple-Instance Problem: a Lazy Learning Approach". In: *International Conference on Machine Learning*. Morgan Kaufmann Publishers. 2000, pp. 1119–1126.

[65] Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. "Multi-instance learning by treating instances as non-iid samples". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 1249–1256. DOI: 10.1145/1553374.1553534.

[66] Jason Farquhar, David Hardoon, Hongying Meng, John S Shawe-taylor, and Sandor Szedmak. "Two view learning: SVM-2K, theory and practice". In: *Advances in neural information processing systems*. 2006, pp. 355–362.

[67] Qiaona Chen and Shiliang Sun. "Hierarchical multi-view fisher discriminant analysis". In: *International Conference on Neural Information Processing*. Springer. 2009, pp. 289–298. DOI: 10.1007/978-3-642-10684-2_32.

[68] Min-Ling Zhang and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning". In: *Pattern recognition* 40.7 (2007), pp. 2038–2048. DOI: 10.1016/j.patcog.2006.12.019.

[69] Amanda Clare and Ross D King. "Knowledge discovery in multi-label phenotype data". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2001, pp. 42–53. DOI: 10.1007/3-540-44794-6_4.

[70] André Elisseeff and Jason Weston. "A kernel method for multi-labelled classification". In: *Advances in neural information processing systems*. 2002, pp. 681–687.

[71] Fadi A Thabtah, Peter Cowling, and Yonghong Peng. "MMAC: A new multi-class, multi-label associative classification approach". In: *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. IEEE. 2004, pp. 217–224. DOI: 10.1109/ICDM.2004.10117.

[72] Jose M Moyano, Eva L Gibaja, Krzysztof J Cios, and Sebastián Ventura. "Review of ensembles of multi-label classifiers: Models, experimental study and prospects". In: *Information Fusion* 44 (2018), pp. 33–45. DOI: 10.1016/j.inffus.2017.12.001.

[73] Concha Bielza, Guangdi Li, and Pedro Larranaga. "Multi-dimensional classification with Bayesian networks". In: *International Journal of Approximate Reasoning* 52.6 (2011), pp. 705–727.

[74] Peter R De Waal and Linda C Van Der Gaag. "Inference and learning in multi-dimensional Bayesian network classifiers". In: *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*. Springer. 2007, pp. 501–511. DOI: 10.1007/978-3-540-75256-1_45.

[75] Ofer Dekel, Yoram Singer, and Christopher D Manning. "Log-linear models for label ranking". In: *Advances in neural information processing systems*. 2004, pp. 497–504.

[76] Shai Shalev-Shwartz and Yoram Singer. "A unified algorithmic approach for efficient online label ranking". In: *Artificial Intelligence and Statistics*. 2007, pp. 452–459.

[77] Alan Julian Izenman. "Reduced-rank regression for the multivariate linear model". In: *Journal of multivariate analysis* 5.2 (1975), pp. 248–264. DOI: 10.1016/0047-259X(75)90042-1.

[78] A Van Der Merwe and JV Zidek. "Multivariate regression analysis and canonical variates". In: *Canadian Journal of Statistics* 8.1 (1980), pp. 27–39. DOI: 10.2307/3314667.

[79] Emmanuel Vazquez and Eric Walter. "Multi-output support vector regression". In: *13th IFAC Symposium on System Identification*. Citeseer. 2003, pp. 1820–1825.

[80] Matilde Sánchez-Fernández, Mario de-Prado-Cumplido, Jerónimo Arenas-García, and Fernando Pérez-Cruz. "SVM multiregression for nonlinear channel estimation in multiple-input multiple-output systems". In: *IEEE transactions on signal processing* 52.8 (2004), pp. 2298–2307. DOI: 10.1109/TSP.2004.831028.

[81] Charles A Micchelli and Massimiliano Pontil. "On learning vector-valued functions". In: *Neural computation* 17.1 (2005), pp. 177–204. DOI: 10.1162/0899766052530802.

[82] Mauricio A Alvarez, Lorenzo Rosasco, and Neil D Lawrence. "Kernels for Vector-Valued Functions: A Review". In: *Foundations and Trends in Machine Learning*. Now Publishers, 2012. DOI: 10.1561/2200000036.

[83] Glenn De'Ath. "Multivariate regression trees: a new technique for modeling species–environment relationships". In: *Ecology* 83.4 (2002), pp. 1105–1117. DOI: 10.1890/0012-9658(2002)083[1105:MRTANT]2.0.CO;2.

[84] Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. "Tree ensembles for predicting structured outputs". In: *Pattern Recognition* 46.3 (2013), pp. 817–833. DOI: 10.1016/j.patcog.2012.09.023.

[85] Mario Costa. "Probabilistic interpretation of feedforward network outputs, with relationships to statistical prediction of ordinal quantities". In: *International journal of neural systems* 7.05 (1996), pp. 627–637. DOI: 10.1142/S0129065796000610.

[86] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. "A neural network approach to ordinal regression". In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE. 2008, pp. 1279–1284. DOI: 10.1109/IJCNN.2008.4633963.

[87] Wan-Yu Deng, Qing-Hua Zheng, Shiguo Lian, Lin Chen, and Xin Wang. "Ordinal extreme learning machine". In: *Neurocomputing* 74.1-3 (2010), pp. 447–456. DOI: 10.1016/j.neucom.2010.08.022.

[88] Javier Sánchez-Monedero, Pedro Antonio Gutiérrez, and Cesar Hervás-Martínez. "Evolutionary ordinal extreme learning machine". In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2013, pp. 500–509. DOI: 10.1007/978-3-642-40846-5_50.

[89] Jaime S Cardoso and Ricardo Sousa. "Classification models with global constraints for ordinal data". In: *2010 Ninth International Conference on Machine Learning and Applications*. IEEE. 2010, pp. 71–77. DOI: 10.1109/ICMLA.2010.18.

[90] Wei Chu and Zoubin Ghahramani. "Gaussian processes for ordinal regression". In: *Journal of machine learning research* 6.Jul (2005), pp. 1019–1041.

[91] Hsuan-Tien Lin and Ling Li. "Combining ordinal preferences by boosting". In: *Proceedings ECML/PKDD 2009 Workshop on Preference Learning*. 2009, pp. 69–83.

[92] Wouter Duivesteijn and Ad Feelders. "Nearest neighbour classification with monotonicity constraints". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2008, pp. 301–316. DOI: 10.1007/978-3-540-87479-9_38.

[93] Rob Potharst and Adrianus Johannes Feelders. "Classification trees for problems with monotonicity constraints". In: *ACM SIGKDD Explorations Newsletter* 4.1 (2002), pp. 1–10. DOI: 10.1145/568574.568577.

[94] Krzysztof Dembczyński, Wojciech Kotłowski, and Roman Słowiński. "Ensemble of decision rules for ordinal classification with monotonicity constraints". In: *International Conference on Rough Sets and Knowledge Technology*. Springer. 2008, pp. 260–267. DOI: 10.1007/978-3-540-79721-0_38.

[95] Jerzy Błaszczyński, Roman Słowiński, and Marcin Szelag. "Sequential covering rule induction algorithm for variable consistency rough set approaches". In: *Information Sciences* 181.5 (2011), pp. 987–1002. DOI: 10.1016/j.ins.2010.10.030.

[96] Joseph Sill. "Monotonic networks". In: *Advances in neural information processing systems*. 1998, pp. 661–667.

[97] Sotiris Kotsiantis, Dimitris Kanellopoulos, and Vasilis Tampakas. "Financial application of multi-instance learning: two greek case studies". In: *Journal of Convergence Information Technology* 5.8 (2010), pp. 42–53.

[98] Massih Amini, Nicolas Usunier, and Cyril Goutte. "Learning from multiple partially observed views-an application to multilingual text categorization". In: *Advances in neural information processing systems*. 2009, pp. 28–36.

[99] Stan Z Li et al. "Statistical learning of multi-view face detection". In: *European Conference on Computer Vision*. Springer. 2002, pp. 67–81. DOI: 10.1007/3-540-47979-1_5.

[100] Sinno Jialin Pan, James T Kwok, Qiang Yang, and Jeffrey Junfeng Pan. "Adaptive localization in a dynamic WiFi environment through multi-view learning". In: *AAAI*. 2007, pp. 1108–1113.

[101] Shiliang Sun and Guoqing Chao. "Multi-View Maximum Entropy Discrimination." In: *IJCAI*. 2013, pp. 1706–1712.

[102] Grigorios Tzortzis and Aristidis Likas. "Kernel-based weighted multi-view clustering". In: *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 675–684. DOI: 10.1109/ICDM.2012.43.

[103] I. Katakis, G. Tsoumakas, and I. Vlahavas. "Multilabel Text Classification for Automated Tag Suggestion". In: *Proc. ECML PKDD08 Discovery Challenge, Antwerp, Belgium*. 2008, pp. 75–83.

[104] M. Boutell, J. Luo, X. Shen, and C. Brown. "Learning multi-label scene classification". In: *Pattern Recognition* 37.9 (2004), pp. 1757–1771. DOI: 10.1016/j.patcog.2004.03.009.

[105] Francisco Charte, Antonio J. Rivera, Maria J. del Jesus, and Francisco Herrera. "QUINTA: A question tagging assistant to improve the answering ratio in electronic forums". In: *EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE*. Sept. 2015, pp. 1–6. DOI: 10.1109/EUROCON.2015.7313677.

[106] S. Diplaris, G. Tsoumakas, P. Mitkas, and I. Vlahavas. "Protein Classification with Multiple Algorithms". In: *Proc. 10th Panhellenic Conference on Informatics, Volos, Greece, PCI05*. 2005, pp. 448–456. DOI: 10.1007/11573036_42.

[107] Ricardo Sousa and João Gama. "Multi-label classification from high-speed data streams with adaptive model rules and random rules". In: *Progress in Artificial Intelligence* 7.3 (Sept. 2018), pp. 177–187. DOI: 10.1007/s13748-018-0142-z.

[108] Khalil Laghmari, Christophe Marsala, and Mohammed Ramdani. "An adapted incremental graded multi-label classification model for recommendation systems". In: *Progress in Artificial Intelligence* 7.1 (Mar. 2018), pp. 15–29. DOI: 10.1007/s13748-017-0133-5.

[109] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. "Multilabel classification via calibrated label ranking". In: *Machine learning* 73.2 (2008), pp. 133–153. DOI: 10.1007/s10994-008-5064-8.

[110] Michael B Eisen, Paul T Spellman, Patrick O Brown, and David Botstein. "Cluster analysis and display of genome-wide expression patterns". In: *Proceedings of the National Academy of Sciences* 95.25 (1998), pp. 14863–14868.

[111] Michael Lyons, Shigeru Akamatsu, Miyuki Kamachi, and Jiro Gyoba. "Coding facial expressions with gabor wavelets". In: *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*. IEEE. 1998, pp. 200–205. DOI: 10.1109/AFGR.1998.670949.

[112] Dragi Kocev, Sašo Džeroski, Matt D White, Graeme R Newell, and Peter Griffioen. "Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition". In: *Ecological Modelling* 220.8 (2009), pp. 1159–1168. DOI: 10.1016/j.ecolmodel.2009.01.037.

[113] Damjan Kuznar, Martin Mozina, and Ivan Bratko. "Curve prediction with kernel regression". In: *Proceedings of the 1st Workshop on Learning from Multi-Label Data*. 2009, pp. 61–68.

[114] Devis Tuia, Jochem Verrelst, Luis Alonso, Fernando Pérez-Cruz, and Gustavo Camps-Valls. "Multioutput support vector regression for remote sensing biophysical parameter estimation". In: *IEEE Geoscience and Remote Sensing Letters* 8.4 (2011), pp. 804–808. DOI: 10.1109/LGRS.2011.2109934.

[115] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. "Feature selection for ordinal text classification". In: *Neural computation* 26.3 (2014), pp. 557–591. DOI: 10.1162/NECO_a_00558.

[116] Qing Tian, Songcan Chen, and Xiaoyang Tan. "Comparative study among three strategies of incorporating spatial structures to ordinal image regression". In: *Neurocomputing* 136 (2014), pp. 152–161. DOI: 10.1016/j.neucom.2014.01.017.

[117] Ralf Bender and Ulrich Grouven. "Ordinal logistic regression in medical research". In: *Journal of the Royal College of physicians of London* 31.5 (1997), pp. 546–551.

[118] Young S Kwon, Ingoo Han, and Kun Chang Lee. "Ordinal pairwise partitioning (OPP) approach to neural networks training in bond rating". In: *Intelligent Systems in Accounting, Finance & Management* 6.1 (1997), pp. 23–40. DOI: 10.1002/(SICI)1099-1174(199703)6:1<23::AID-ISAF113>3.0.CO;2-4.

[119] Kuang-Yu Chang, Chu-Song Chen, and Yi-Ping Hung. "Ordinal hyperplanes ranker with cost sensitivities for age estimation". In: *Computer vision and pattern recognition (cvpr), 2011 ieee conference on*. IEEE. 2011, pp. 585–592. DOI: 10.1109/CVPR.2011.5995437.

[120] Salvatore Greco, Benedetto Matarazzo, and Roman Słowiński. "Rough set approach to customer satisfaction analysis". In: *International Conference on Rough Sets and Current Trends in Computing*. Springer. 2006, pp. 284–295. DOI: 10.1007/11908029_31.

[121] Salvatore Greco, Benedetto Matarazzo, and Roman Slowinski. "A new rough set approach to evaluation of bankruptcy risk". In: *Operational tools in the management of financial risks*. Springer, 1998, pp. 121–136. DOI: 10.1007/978-1-4615-5495-0_8.

[122] Young U Ryu, Ramaswamy Chandrasekaran, and Varghese S Jacob. “Breast cancer prediction using the isotonic separation technique”. In: *European Journal of Operational Research* 181.2 (2007), pp. 842–854. DOI: 10.1016/j.ejor.2006.06.031.

[123] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S Yu. “Building text classifiers using positive and unlabeled examples”. In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE. 2003, pp. 179–186. DOI: 10.1109/ICDM.2003.1250918.

[124] Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. “Learning structured prediction models: A large margin approach”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 896–903. DOI: 10.1145/1102351.1102464.

An analysis on the use of autoencoders for representation learning

IV

Source

David Charte, Francisco Charte, Maria J del Jesus, and Francisco Herrera. "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges". In: *Neurocomputing* 404 (2020), pp. 93–107. DOI: 10.1016/j.neucom.2020.04.057

Abstract

In many machine learning tasks, learning a good representation of the data can be the key to building a well-performant solution. This is because most learning algorithms operate with the features in order to find models for the data. For instance, classification performance can improve if the data is mapped to a space where classes are easily separated, and regression can be facilitated by finding a manifold of data in the feature space. As a general rule, features are transformed by means of statistical methods such as principal component analysis, or manifold learning techniques such as Isomap or locally linear embedding. From a plethora of representation learning methods, one of the most versatile tools is the autoencoder. In this paper we aim to demonstrate how to influence its learned representations to achieve the desired learning behavior. To this end, we present a series of learning tasks: data embedding for visualization, image denoising, semantic hashing, detection of abnormal behaviors and instance generation. We model them from the representation learning perspective, following the state of the art methodologies in each field. A solution is proposed for each task employing autoencoders as the only learning method. The theoretical developments are put into practice using a selection of datasets for the different problems and implementing each solution, followed by a discussion of the results in each case study and a brief explanation of other six learning applications. We also explore the current challenges and approaches to explainability in the context of autoencoders. All of this helps conclude that, thanks to alterations in their structure as well as their objective function, autoencoders may be the core of a possible

solution to many problems which can be modeled as a transformation of the feature space.

Keywords

representation learning - autoencoders - deep learning - feature extraction

IV.1 Introduction

Creating new representations of data is a fundamental task in most machine learning tasks. First off, certain types of problems that require a classifier or a regressor will certainly benefit from transformations of the features which facilitate their work [2]. In addition to this, there exists a variety of problems whose solution relies strongly on finding an appropriate representation of the data. Although the use of representation learning techniques is mainly used as a complement to other learners in the former case, in the latter one these methods become the focus. This work highlights some of these situations, with specific applications that can be modeled as representation learning problems.

[2]: Domingos (2012), "A Few Useful Things to Know About Machine Learning"

The features that are used as input conform one of the most important factors when building machine learning models. When the training set contains intact data from its collection or measurements, it may not be ready for treatment yet. Instead, it is common for data to be expressed with redundant or uninformative variables and for it to include some level of noise. These and other obstacles presented by the data [3] are the reason why most of the manual work of building machine learning models is spent in the preprocessing stage [4].

[3]: Lorena et al. (2019), "How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity"

[4]: García et al. (2015), *Data preprocessing in data mining*

The success of a classifier, a regressor or other models will greatly depend on the quality of the features it can learn from. For instance, decision trees, regardless of whether the task is classification or regression, attempt to find the most informative variables to branch at each step [5]; support vector machines calculate the hyperplane that best separates classes in a feature space originating from specific transformations of the original one [5], and k-means clustering computes distances among pairs of instances and thus depends strongly on the input domain [6]. As a result, it is of vital importance that the features provided to these learners are useful and as independent as possible.

[5]: Kotsiantis et al. (2007), "Supervised machine learning: A review of classification techniques"

[5]: Kotsiantis et al. (2007), "Supervised machine learning: A review of classification techniques"

[6]: Jain et al. (1999), "Data Clustering: A Review"

However, finding alternative representations for data is not only a medium to build classification and regression models, but it may be an end in itself in many applications. For example, finding compact binary codes that represent text documents [7], compressing signals to a lower resolution without losing information [8], transforming the problem domain to a different one [9], or producing filtered versions of images with less distortions [10].

[7]: Salakhutdinov et al. (2009), "Semantic hashing"

[8]: Theis et al. (2017), "Lossy image compression with compressive autoencoders"

[9]: Deng et al. (2013), "Sparse autoencoder-based feature transfer learning for speech emotion recognition"

[10]: Xie et al. (2012), "Image denoising and inpainting with deep neural networks"

Learning representations usually consists in feature engineering [2] or feature extraction [11], depending on whether new features are computed manually by human intervention (either by selection [12] or simple arithmetic operations) or they are generated, evaluated and selected by the machine. Feature engineering leverages expert knowledge and human creativity in order to select features and operate with them in a way that results in a new feature set which seems appropriate for predictors to work with. Nowadays there exist many automatic approaches to feature learning, which relieve users from the tedious task of engineering new features [13]. These methods range from probabilistic to topological and from shallow to deep: principal component analysis [14], Isomap [15], locally linear embedding [16] and Laplacian eigenmaps [17], among others.

[2]: Domingos (2012), "A Few Useful Things to Know About Machine Learning"

[11]: Guyon et al. (2008), *Feature extraction: foundations and applications*

[12]: Dash et al. (1997), "Feature selection for classification"

[13]: Bengio et al. (2013), "Representation learning: A review and new perspectives"

[14]: Jolliffe (1986), *Principal component analysis*

[15]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

[16]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

[17]: Belkin et al. (2003), "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation"

With the introduction of deep neural networks, the representation learning stage became integrated within the predictors themselves [18]. These techniques iteratively optimize the classification performance by modifying the weights in several layers of individual neurons which compute a hierarchy of abstractions over the original data. For this purpose, the backpropagation algorithm [19] allows to efficiently accumulate gradients along the network, so that an optimizer such as Stochastic Gradient Descent [20] or one of its derivatives [21–24] may compute each weight update. Since neural networks can be structured as needed for each kind of problem, they are able to function as standalone feature learners as well. This is the case of autoencoders (AEs) [25], neural architectures whose objective is to find the best representation for the data according to the criterium defined by their loss function.

[18]: LeCun et al. (2015), "Deep learning"

[19]: Rumelhart et al. (1988), "Learning representations by back-propagating errors"

[20]: Robbins et al. (1951), "A stochastic approximation method"

[21]: Duchi et al. (2011), "Adaptive subgradient methods for online learning and stochastic optimization"

[22]: Zeiler (2012), "Adadelta: an adaptive learning rate method"

[23]: Kingma et al. (2015), "Adam: A method for stochastic optimization"

[24]: Tieleman et al. (2012), "Lecture 6.5-RMSProp"

The objective of this paper is to analyze how AEs can serve as the main basis for solving a wide variety of learning tasks and demonstrate this with concrete applications and experimental results. Throughout the paper, we examine several case studies that expose the adaptability of AEs to these problems.

- First, an example of data embedding onto a very low dimensional space for visualization and exploratory analysis.

[25]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

- Then, a case where noisy signals are to be repaired by the model.
- Later, a different example where very high dimensional sparse data, such as text documents, is to be compressed onto compact binary codes in a semantic way.
- Additionally, we study anomaly detection, the situation where abnormal patterns are to be detected in sequences but no anomalies are available to learn from.
- As a last case study, we propose the generation of new instances which do not belong to the training set.

Other applications are also briefly discussed: image superresolution, image compression, transfer learning, human pose recovery and recommender systems.

As a starting point, we provide the reader with the necessary background knowledge about the field of representation learning, as well as a summary of the main features of AEs that make them a good candidate model to solve the different problems later approached. The solutions to these tasks using AEs as the only automatic learner highlight their potential and flexibility as feature extraction techniques.

[26]: Arrieta et al. (2020), "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI"

Following the current increase in search for developing explainable models [26], the main approaches for obtaining interpretable predictions are summarized, finding that quality features can be the key to explainable solutions. AE models which can build helpful features are also highlighted.

The rest of this paper is structured as follows. Section IV.2 describes the background of the problems and techniques above introduced. Section IV.3 details the inner workings of AEs. Section IV.4 further develops on several case studies where AEs resolve feature learning tasks and outlines other existing learning applications, and Section IV.5 describes the current state of the art in explainable AI and how AEs are involved. Lastly, Section IV.6 concludes the text.

IV.2 Background: feature learning and deep representation learning

This section explains some well-known methods that can extract features from data. Afterwards, it introduces deep learning techniques.

Classical feature learning methods

Traditionally, feature extraction methods have been developed with linear as well as nonlinear transformations of the variables [11]. They can be considered nonconvex or convex, according to whether the objective function presents local optima or not, respectively [27]. Many of these techniques perform unsupervised learning, but others are supervised [28–30] or even semi-supervised [31]. Next, a summary of typical feature learning methods is provided.

[11]: Guyon et al. (2008), *Feature extraction: foundations and applications*

[27]: Van Der Maaten et al. (2009), *Dimensionality reduction: a comparative review*

[28]: Fisher (1936), "The use of multiple measurements in taxonomic problems"

[29]: Zhao et al. (2006), "Local structure based supervised feature extraction"

[30]: Makhzani et al. (2016), "Adversarial autoencoders"

[31]: Zhao et al. (2008), "Locality sensitive semi-supervised feature selection"

Linear methods The most common linear feature extraction methods are the following. Principal component analysis (PCA) consists in extracting successive variables or *principal components* with maximum variance while being uncorrelated with the previous components. It is a statistical technique developed geometrically by Pearson [32] and algebraically by Hotelling [33], but an analytical derivation can be found in [14]. Factor analysis [34] is a similar procedure to PCA which considers a set of latent variables or *factors* that are not observed but are linearly combined to produce the final variables. Linear discriminant analysis [28] is a supervised statistical technique which attempts to find linear combinations of features to project samples onto new coordinates that best discriminate classes, albeit making some assumptions about the distribution of the data.

[32]: Pearson (1901), "LIII. On lines and planes of closest fit to systems of points in space"

[33]: Hotelling (1933), "Analysis of a complex of statistical variables into principal components"

[14]: Jolliffe (1986), *Principal component analysis*

[34]: Jolliffe (1986), "Principal Component Analysis and Factor Analysis"

[28]: Fisher (1936), "The use of multiple measurements in taxonomic problems"

Nonlinear methods Some well known nonlinear approaches to feature extraction are kernel PCA, restricted Boltzmann machines and manifold learning methods. Kernel PCA [35] extends PCA to nonlinear combinations of features by projecting samples onto higher-dimensional spaces and using the kernel trick [36]. Restricted Boltzmann machines are undirected graphical probabilistic models, also known as harmoniums [37], with one visible layer and one hidden layer that acts as the set of extracted features. They can be trained using the contrastive divergence algorithm [38]. Many nonlinear feature learning methods attempt to find coordinates for a lower dimensional structure embedded in the original features, namely, a manifold. Multidimensional scaling (MDS) is one of the first techniques that can be considered manifold learning, as its objective is projecting samples in a low-dimensional space while translating as much information of pairwise distances as possible. There are several variants of MDS, one of them is Sammon mapping [39], which improves on MDS by using a different cost function which stresses large distances similarly to small ones. Isomap [15] is a more recent extension of MDS which looks for the coordinates that

[35]: Schölkopf et al. (1998), "Nonlinear component analysis as a kernel eigenvalue problem"

[36]: Schölkopf (2001), "Statistical Learning and Kernel Methods"

[37]: Smolensky (1986), *Information processing in dynamical systems: Foundations of harmony theory*

[38]: Hinton (2002), "Training Products of Experts by Minimizing Contrastive Divergence"

[39]: Sammon (1969), "A nonlinear mapping for data structure analysis"

[15]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances (the length of the shortest path that connects two points in the manifold). Locally Linear Embedding [16] also seeks a manifold which preserves neighbors but, in order to maintain the local structure, it linearly reconstructs each point from its neighbors. Laplacian eigenmaps [17] is a procedure that builds a graph based on the neighborhood structure of the data, and from it a weight matrix whose eigenvectors can be used to compute new coordinates for each point.

[16]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

[17]: Belkin et al. (2003), "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation"

Deep representation learning

Deep learning architectures are hierarchies of abstractions of the input feature space and, as such, they compute several transformations of the features before reaching a response. In some cases, these can be seen as learned representations, since they must be able to capture the relevant information from each instance in order to output an accurate result. This effect can be observed especially in convolutional neural network classifiers, which are usually split into a feature extraction component formed by convolutional layers and a decision module composed by fully connected layers [40]. Apart from neural networks with other objectives such as supervised classification or regression, there have been different approaches to shallow as well as deep neural structures for unsupervised feature learning [41], such as self-organizing Kohonen maps [42, 43], predictability minimization [44], restricted Boltzmann machines [45], deep belief networks [46–48] and AEs [49, 50]. There have been many instances of these unsupervised techniques being used to either pre-train or provide feature transformations for supervised models [51].

AEs are probably the most versatile unsupervised neural network models. They essentially combine some kind of bottleneck or restriction in the learned data representations with the objective of reconstructing and repairing the original input from that representation [25]. There are several ways to impose restrictions that produce interesting representations, and the reconstruction objective will cause the network to retain all invariant feature information along its weights, so that the representation or encoding holds mainly instance-specific traits. For example, undercomplete AEs project inputs into lower-dimensional encodings, sparse AEs obtain representations with very few activated neurons, and denoising AEs attempt to repair partially corrupted data.

[40]: Krizhevsky et al. (2012), "Imagenet classification with deep convolutional neural networks"

[41]: Schmidhuber (2015), "Deep Learning in Neural Networks: An Overview"

[42]: Kohonen (1990), "The self-organizing map"

[43]: Koikkalainen et al. (1990), "Self-organizing hierarchical feature maps"

[44]: Schmidhuber et al. (1996), "Semi-linear predictability minimization produces well-known feature detectors"

[45]: Goodfellow et al. (2016), "Deep Learning"

[46]: Hinton et al. (2006), "Reducing the dimensionality of data with neural networks"

[47]: Zeng et al. (2016), "Deep belief networks for quantitative analysis of a gold immunochromatographic strip"

[48]: Zeng et al. (2019), "An improved particle filter with a novel hybrid proposal distribution for quantitative analysis of gold immunochromatographic strips"

[49]: Kramer (1991), "Nonlinear principal component analysis using autoassociative neural networks"

[50]: Oja (1991), "Data compression, feature extraction, and autoassociation in feedforward neural networks"

[51]: Bengio (2012), "Deep learning of representations for unsupervised and transfer learning"

[25]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

Their versatility is demonstrated by the amount of applications AEs have and their diversity. Across the rest of this work, we focus on certain applications of representation learning that are solved with AEs and we analyze how each model is built and trained.

IV.3 Autoencoder fundamentals

AEs are neural network structures designed with the purpose of learning new features. Throughout the following subsections, their main characteristics and differentiating aspects are outlined, and some ways to influence the encoded variables are discussed.

Origin and essentials of autoencoders

AEs were originally conceieved as a way of initializing neural networks [52] and continued fulfilling that purpose for some time, serving as a starting point for training of deep networks as well [53]. Over the last years, other applications for AEs have been emerging and at the same time other approaches to neural network training and regularization have succeeded over AEs [54, 55]. As a consequence, the common uses for AEs have shifted from helping train other neural networks to other applications of their own.

[52]: Ballard (1987), "Modular Learning in Neural Networks"

[53]: Hinton et al. (2006), "A fast learning algorithm for deep belief nets"

[54]: Glorot et al. (2011), "Deep sparse rectifier neural networks"

[55]: Srivastava et al. (2014), "Dropout: a simple way to prevent neural networks from overfitting"

In general, the training process required to learn an AE can be unsupervised, that is, it does not need labels or class information in order to generate a model for the data. Instead, it extracts useful information from each instance by feeding its feature vector through some transformations which impose a bottleneck or restriction on the possible representations it can compute. Then, the representation is mapped to the original feature space through a similar set of transformations, and the AE is evaluated according to the fidelity of the reconstruction. This feedback allows to modify the parameters iteratively until convergence is reached.

AEs take the form of a neural network with at least one hidden layer and two components, an encoder and a decoder, which are connected by the coding layer [25]. These components are usually symmetric in layer shapes to each other, especially if they are implemented as fully connected neural networks. In certain occasions, even the weights of each layer in the decoder are tied to the corresponding layer in the encoder. In general terms, however, it suffices with the input layer of the encoding and the output layer having the same shape. Fig. IV.1 shows how the architecture of an AE may look like.

[25]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

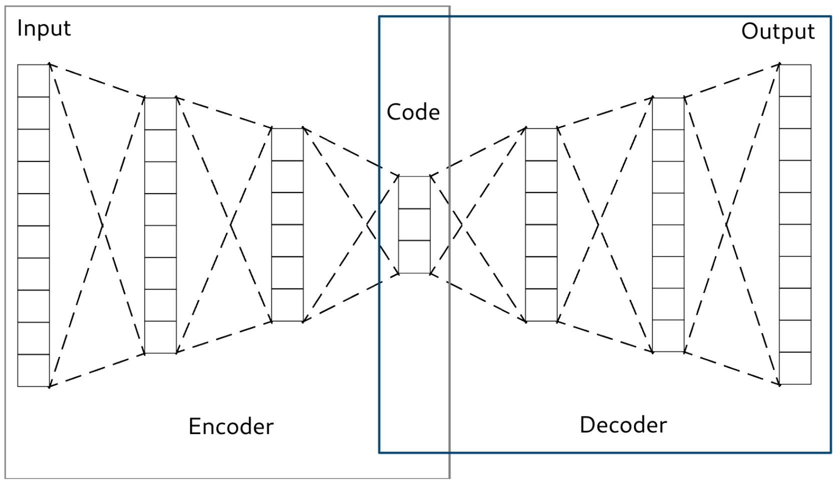


Figure IV.1: Illustration of the general structure of a basic AE: an encoder and a decoder connected by the encoding layer

Table IV.1: Intepretation of symbols used in the formulae

Symbol	Interpretation
θ	Full set of parameters of the AE (weights and biases)
$\mathcal{X}$	Set of input instances
$\mathfrak{X}$	Set of instances in encoding space
n	Dimension of input space
k	Dimension of encoding space
f	Encoder mapping
g	Decoder mapping
d	Distance function in input space
r	Regularization function

In summary, an AE can be seen as the composition of an encoding map f which projects inputs onto a different feature space, and a decoding map g which operates inversely (see Table IV.1 for the meaning of all symbols used below). The main objective of the AE is to recover as much information as possible of the original input, so it will attempt to minimize a distance between the inputs and the outputs:

$$\min_{\theta} \sum_{x \in \mathcal{X}} d(x, g_\theta(f_\theta(x))) \tag{IV.1}$$

The distance function d used in the loss function is usually either the mean squared error, see Eq. (IV.2), or the cross entropy, shown in Eq. (IV.3). In the first case, data may not be normalized and the output units should use an unbounded activation function. For a cross entropy loss, each input and output variable is modeled as following a Bernoulli distribution, so data should be scaled to the $[0, 1]$ interval and output units could make use of a sigmoid activation.

The mean squared error for an input x and output x' of length n is defined as:

$$d(x, x') = \frac{1}{n} \sum_{i=1}^{n} (x_i - x'_i)^2 \tag{IV.2}$$

Similarly, the binary cross entropy for the same input and output is computed as:

$$d(x, x') = -(x \bullet \log(x') + (1 - x) \bullet \log(1 - x')), \tag{IV.3}$$

where $\bullet$ denotes element-wise product and all other operations are also performed element-wise.

Modeling the coding layer

The main objective of the AE (Eq. IV.1) only promotes faithful reconstructions without explicitly considering any aspect about the codes used. This can be enough in many cases where the codes are low dimensional and they can capture only the relevant information of the instances just by training to reconstruct accurately. Notwithstanding, there are situations that require considering a more general case of the objective, which allows penalizing certain behaviors of the encoding found by the network, or even the values of the parameters themselves (Eq. IV.4).

$$\min_{\theta} \sum_{x \in \mathcal{X}} d(x, g_\theta(f_\theta(x))) + r_1(f_\theta(\mathcal{X})) + r_2(\theta) \tag{IV.4}$$

A straightforward example of this kind of restrictions is the sparse AE [56, 57], which adds a penalty for high activation rates in the neurons of the code layer (Eqs. IV.5 and IV.6):

[56]: Lee et al. (2008), "Sparse deep belief net model for visual area V2"
[57]: Ng et al. (2011), "Sparse autoencoder"

$$r(\mathcal{Z}) = \sum_{j=1}^{k} (\rho - \rho_j)^2, \text{ or} \tag{IV.5}$$

$$r(\mathcal{Z}) = \sum_{j=1}^{k} \rho \log \frac{\rho}{\rho_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \rho_j}, \tag{IV.6}$$

where $\rho_j = \frac{1}{|\mathcal{Z}|} \sum_{z \in \mathcal{Z}} z_j$ is the average activation vector, k is the length of the code and ρ is the desired activation rate.

Other, more sophisticated variations on the AE with different penalties are the contractive AE [58, 59], which promotes finding and preserving any local structure from the original feature space, and the variational AE [60], which uses a penalty to impose a distribution to the codes computed by the encoder.

[58]: Rifai et al. (2011), "Contractive auto-encoders"
[59]: Rifai et al. (2011), "Higher order contractive auto-encoder"
[60]: Kingma et al. (2013), "Auto-encoding variational bayes"

Penalties on the codes are not, however, the only way of incentivizing a behavior on the encoder mapping. Denoising AEs [61, 62] establish a slightly different criterion to evaluate the performance of the reconstruction: the network must be able to repair any noise or corruption from the input. Robust AEs [63] use another objective function, correntropy [64], which has a similar effect in repairing several kinds of noise from the input data.

[61]: Vincent et al. (2008), "Extracting and composing robust features with denoising autoencoders"
[62]: Vincent et al. (2010), "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion"
[63]: Qi et al. (2014), "Robust feature learning by stacked autoencoder with maximum correntropy criterion"
[64]: Liu et al. (2006), "Correntropy"

Evaluation metrics

The quality of learned features can be evaluated by the model's ability to project instances back to the original feature space. For this purpose, regression metrics can be used. Some common metrics which serve to assess the usefulness of the learned features are the following, where x is the original feature vector and x' is the reconstruction, mapped from the encoding space back onto the input space:

- Mean squared error (Eq. IV.2) and root mean squared error:

$$\mathrm{RMSE}(x, x') = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - x'_i\right)^2}$$

- Mean absolute error:

$$\mathrm{MAE}(x, x') = \frac{1}{n}\sum_{i=1}^{n}\left|x_i - x'_i\right|$$

- Mean absolute percentage error

$$\mathrm{MAPE}(x, x') = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{x_i - x'_i}{x_i}\right|$$

In certain cases, the encoded features can also be evaluated independently from the original features, by assessing their quality with respect to their complexity, class separability and overlap [3]. This usually requires that data belongs to a classification problem so that a class is defined for each instance.

[3]: Lorena et al. (2019), "How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity"

Beyond unsupervised autoencoders

Although the objective of an AE usually does not involve direct prediction of labels, it can sometimes learn from classified examples. The most straightforward way to introduce class information into the AE is to modify the loss function so it propagates different errors according to the class of each instance. For example, we could weight each class differently. Assuming the classes are binary, dividing the dataset into $\mathcal{X}^+$ for positive instances and $\mathcal{X}^-$ for negative ones, and α is a parameter in $[0, 1]$, the objective in Eq. (IV.7)

$$\min_\theta (1-\alpha) \sum_{x\in\mathcal{X}^-} d(x, g_\theta(f_\theta(x))) + \alpha \sum_{x\in\mathcal{X}^+} d(x, g_\theta(f_\theta(x))) \qquad \text{(IV.7)}$$

would give more importance to reconstructing one of the classes, which may help if the aim is to find a manifold for that class and the other one is less relevant.

Several uses of label information can be found in the proposal of the adversarial AE [30]. This AE has a similar behavior to the variational AE in that it also forces the codes to follow a given distribution. Instead of using just a loss penalty, it adds a generator which samples the distribution, and a discriminator which attempts to distinguish distribution samples from codes belonging to actual instances, analogous to a generative adversarial network [65]. The label information can be used then to locate each label in a region of the distribution, by feeding labels as well as codes to the discriminator. Alternatively, labels can be feeded to the decoder, which causes the codes to discard label information and instead model style in the data.

[30]: Makhzani et al. (2016), "Adversarial autoencoders"

[65]: Goodfellow et al. (2014), "Generative adversarial nets"

Another step forward in introducing label information in AEs would be for them to be able to predict labels as well. Some work has been already done along these lines, by training an encoder and decoder simultaneously to reconstruct and to produce codes as similar as possible to the labels in a one-hot format [66].

[66]: Zhuang et al. (2015), "Supervised representation learning: Transfer learning with deep autoencoders"

IV.4 Learning task case studies

The following subsections detail several real examples of application of AEs: embedding data onto a very low-dimensional space for visualization purposes, reducing the noise in images, computing semantic hashes for large text documents, finding anomalous behaviors in sequences and generating new instances outside the training set. For each application, a relevant dataset has been selected and a model has been specifically designed to solve the problem. The basic traits of all chosen datasets can be found in Table IV.2.

The models described below are each associated to a diagram describing the layer structure of the corresponding AE and the purpose of each layer. Please refer to Fig. IV.2 for an example of how each model is detailed.

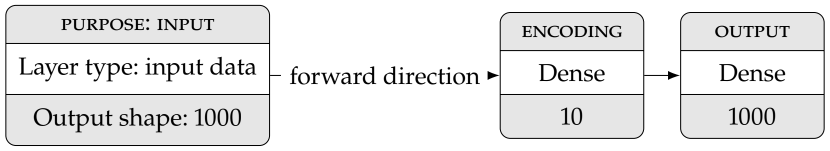


Figure IV.2: Example AE architecture. Each block represents a layer and is splitted into three parts: the meaning or purpose of the layer, the type of operation performed and its output shape (size of each dimension).

All examples have been implemented and executed employing the following setup: Tensorflow [70] 1.14.0 and Keras [71] 2.2.4 on top of Python 3.7 and R 3.6, running on an Intel Core i5-8400 CPU and a NVIDIA GeForce RTX 2060 GPU. The associated software can be found at the following GitHub repository: `https://github.com/ari-dasci/autoencoder-case-studies/`.

[70]: Abadi et al. (2016), "Tensorflow: A system for large-scale machine learning"

[71]: Chollet et al. (2022), *Keras API Documentation*

Data visualization

Most of the data collected nowadays, either from industries or from the web, is high-dimensional. Visualization techniques can help its interpretability, but the data generally needs to be summarized for this purpose. Traditionally, an alternative representation would be a subset of its features or its principal components [72]. An AE, however, is able to automatically compute a representation that fits each dataset. This representation can be 2 or 3-dimensional if the AE is configured conveniently [73], or if another embedding technique (such as t-SNE [74]) is used after a higher-dimensional encoding.

[72]: Jolliffe (2011), "Principal Component Analysis"

[73]: Yu et al. (2013), "Embedding with autoencoder regularization"

[74]: Maaten et al. (2008), "Visualizing data using t-SNE"

In particular, if our dataset consists of instances (x, y) where x is a feature vector and y is its associated label, we can use a training subset to learn an autoencoder model with an encoding $f : \mathbb{R}^n \to \mathbb{R}^2$ resulting of the composition of the hidden layers up to the code layer. Then, encoded examples can be colored in a scatter plot according to their class.

Although a simple AE could fulfill the embedding task, it can be convenient to restrict or modify its behavior so as to influence the

Table IV.2: Main traits of datasets used for the experiments

Dataset	Application	Input features	Training examples	Test examples
CPU Activity	Visualization	21	6553	1639
Satellite image	Visualization	36	5142	1288
STL10 [67]	Noise reduction	$96 \times 96 \times 3$	5000	8000
Bibtex[68]	Semantic hashing	1836	5916	1479
UNSW-NB15[69]	Anomaly detection	187	37000	175341
AT&T faces	Instance generation	64×64	400	-

projection to the embedding space, in a way that improves how the populated regions in the original space are modeled. Along these lines, there are several approaches: denoising criteria [61], contractive regularizations [58] and embedding regularizations [73].

[61]: Vincent et al. (2008), "Extracting and composing robust features with denoising autoencoders"

[58]: Rifai et al. (2011), "Contractive auto-encoders"

[73]: Yu et al. (2013), "Embedding with autoencoder regularization"

Denoising criterion A denoising AE [61] trains, as briefly explained in Section IV.3, by reconstructing partially corrupted inputs. In order to do this, a corruption or noise function introduces alterations on the input data: for example, a Gaussian noise $\xi \sim N(0, \sigma)$ would be used to produce the input $\nu(x) = x + \xi$. The reconstruction error is now computed as $\sum_{x\in\mathcal{X}} d(x, g(f(\nu(x))))$. During the training process, the AE is forced to distinguish useful information from mere perturbations of the data. If the instances lie on a manifold in the original feature space, this can effectively train the AE to "push back" instances to the manifold by discarding small displacements from it. This can remove noise in the inputs as well as reconstruct some missing values if inputs are just an estimation [10, 75]. As a result, the encoding can serve as a set of coordinates for the manifold.

[61]: Vincent et al. (2008), "Extracting and composing robust features with denoising autoencoders"

[10]: Xie et al. (2012), "Image denoising and inpainting with deep neural networks"

[75]: Li et al. (2015), "Feature learning from incomplete EEG with denoising autoencoder"

Contractive regularization The contractive AE [58] uses an additional penalty in the training objective which promotes local invariance to displacements in many directions around the training samples, i.e., it is less sensitive to small perturbations especially in directions that lead outside the manifold. The penalty consists in the squared Frobenius norm of the Jacobian matrix of the encoder, that is, the sum of the squares of all first-order partial derivatives applied to all inputs: $\sum_x \|J_f(x)\|^2$. This can be seen as a generalization of L2 weight decay to the case where the encoder is nonlinear. This regularization favors encodings where all dimensions are contracted, but the reconstruction error prevents the AE from contracting dimensions along the manifold.

[58]: Rifai et al. (2011), "Contractive auto-encoders"

Embedding regularization An alternative objective function for AEs can be the same loss function from other embedding techniques. This is the idea behind embeddings with AE regularization [73], which combines the reconstruction error with one of several possible embedding loss functions coming from Laplacian eigenmaps [17], multidimensional scaling [76] and margin-based embedding [77]. These loss functions evaluate the embedding by taking pairs of instances, and the AE is adapted the same way, by computing the embedding loss across all pairs of instances and the reconstruction loss across all instances.

[73]: Yu et al. (2013), "Embedding with autoencoder regularization"

[17]: Belkin et al. (2003), "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation"

[76]: Torgerson (1952), "Multidimensional scaling: I. Theory and method"

[77]: Hadsell et al. (2006), "Dimensionality reduction by learning an invariant mapping"

For the purposes of demonstrating the capacity of AEs to find manifolds and appropriate embeddings, we have selected a regression dataset, CPU activity*, and a classification dataset, Satellite image†. The AE used to find embeddings is the contractive AE. AEs for both datasets have been designed using the same criteria: three hidden layers, the encoding layer having 2 variables and the rest having as much variables as needed so that the compression ratio from the input to the first hidden layer is the same than from the hidden layer to the encoding layer. The resulting architectures are detailed in Fig. IV.3. The AEs have found the projections shown in Fig. IV.4, where the label of each instance is used to color each point. Notice that the AEs have trained without the respective target variables, but there appears to be some degree of separability of classes and different values of the regression variable in each graph.

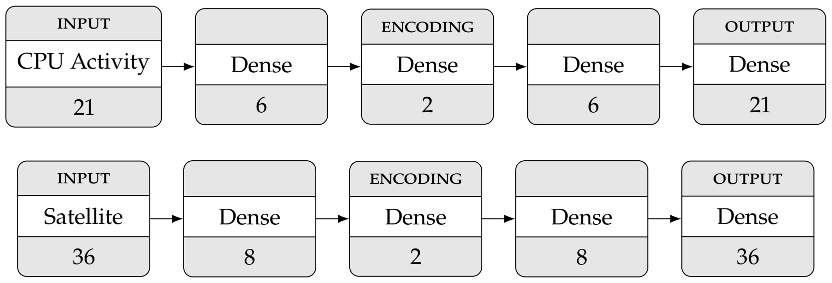


Figure IV.3: AE architectures for visualization

In order to verify to a certain degree that these embeddings, in addition to producing meaningful visualizations, contain the necessary information about the data, the mean squared error between each instance and its reconstruction through the AE can be computed. As a reference for comparison purposes, the same reconstruction error can be computed from the two first principal components of the data and from the encoding found by a basic AE. Table IV.3 holds these results, which are very favourable to the contractive AE, since the error is lower in every case. The difference among both AEs is small, but it serves to deduce that the contractive penalty in the AE does not hinder the reconstruction objective, instead it helps obtain useful low-dimensional embeddings.

Noise reduction

Similar to searching for interesting representations of data in the encodings of an AE, we can look for a reconstruction that adds value to the input data. One way an AE can help with this is to remove

* CPU activity dataset is available at `https://www.openml.org/d/573`.
† Satellite image dataset can be found at `https://www.openml.org/d/294`.

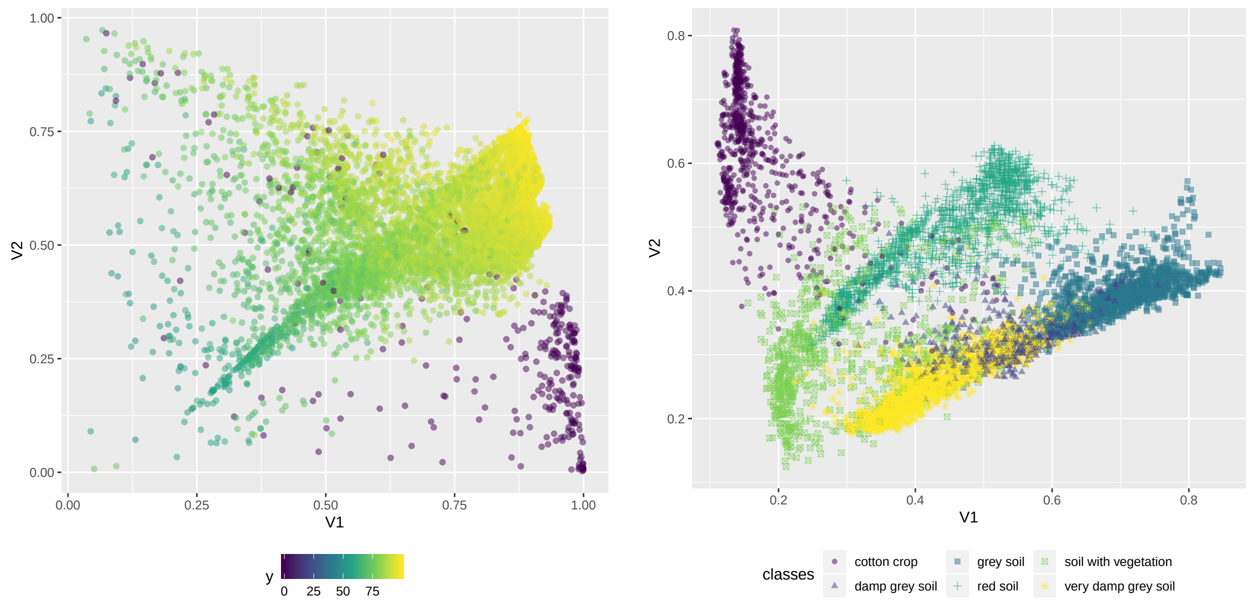


Figure IV.4: Embeddings learned by an unsupervised contractive AE. The top image shows the projection of the CPU Activity dataset where each point has been shaded according to the level of user activity. The bottom image displays the projected samples of the Satellite Image dataset, each one colored according to its class.

	Mean squared error			
	CPU Activity		Satellite	
Method	train	test	train	test
PCA	0.5577	0.5097	0.1475	0.1483
Basic AE	0.5238	0.4729	0.1136	0.1160
Contractive AE	**0.5053**	**0.4546**	**0.1132**	**0.1157**

Table IV.3: Mean squared error comparison between the reconstructions of a contractive AE with a 2-variable encoding and the projections to the original feature space from the two principal components of the data. Lower values are better.

noise from its inputs. This is especially useful when dealing with images [10], sound [78] and other kinds of signals [79], since capture methods usually may introduce some noise and it would be desirable to have a clearer and sharper output.

[10]: Xie et al. (2012), "Image denoising and inpainting with deep neural networks"
[78]: Lu et al. (2013), "Speech enhancement based on deep denoising autoencoder"
[79]: Xiong et al. (2016), "ECG signal enhancement based on improved denoising auto-encoder"

In general, an AE can be trained to be resilient to input perturbations with a mere random additive noise at the input. Throughout the optimization stage, the AE only takes as input partially corrupted versions of the training examples and attempts to reconstruct the original ones. Once trained, this AE does not necessarily expect more noisy data, but instead it will have learned to be robust against small changes in its inputs. This type of AE is usually called a denoising AE, and performs well in many scenarios that do not necessarily involve treatment of noisy data [62].

[62]: Vincent et al. (2010), "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion"

In this case, nonetheless, the goal is to eliminate potential perturbations in the inputs. Unlike a generic noise reduction filter, which will perform similar operations no matter what data it receives, a denoising AE can be fitted to a specific training set and may thus be more reliable with different kinds of data. More formally, we consider a noise function ν, which generates the corrupted data that the autoencoder trains with to minimize

$$\sum_{x\in\mathcal{X}} d(x, g(f(\nu(x)))) \,.$$

The following are some possible noise functions that may be applied:

- $\nu(x) = x + \xi$ where ξ is sampled from a Gaussian distribution with small variance
- $\nu(x) = x + \xi'$ where ξ' is sampled from a Cauchy distribution with small scale
- $\nu(x) = \begin{cases} 0 & \text{with low probability} \\ x & \text{otherwise} \end{cases}$
- $\nu(x) = \begin{cases} 0 & \text{with low probability} \\ 1 & \text{with low probability} \\ x & \text{otherwise} \end{cases}$

Notice that the Gaussian and Cauchy distributions will usually induce small changes to most inputs, while the zero and zero-one noises will leave most values intact but the change in the corrupted ones will be more drastic. Thus, for a given application, a specific type of corruption function can be selected so that it fits best to the types of noise the samples could have.

When using denoising AEs, it is also convenient to adapt the type of layers used to the kind of data. For instance, a convolutional AE would be best for noisy images, and an LSTM AE for corrupted signals or sequences. Fig. IV.5 details a possible encoder-decoder structure for a denoising AE which uses convolutional layers in the encoding phase as well as deconvolution operations during decodification.

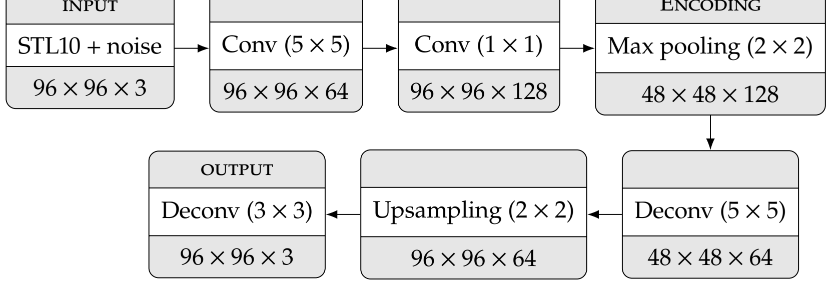


Figure IV.5: Denoising AE architecture for noise reduction

Images	Mean squared error	Noise reduction
Reference	0	100%
Noisy	1656.08 ± 696.31	0%
Basic AE	576.68 ± 156.53	62.14% ± 9.54
Denoising AE	**159.74** ± 74.55	**88.94%** ± 6.38

Table IV.4: Summary of results for noise reduction (average values and standard deviations are provided). Original images without noise are the reference for measuring the mean squared error, and the noise reduction is computed for each image as the percentage decrease in this error. Images are represented by their RGB values from 0 to 255.

When this AE is trained with data from the STL10 dataset [67], a subset of the ImageNet dataset, the objective function will force it to configure its weights so that input noise is reduced along the network. The noise used in this case has been zeros with a probability of 0.1. The test images measure 96x96 pixels and have also been corrupted with around 10% of noisy values, which can affect any color channel, so each pixel has a 30% likelihood of having any of its 3 values altered. The AE was trained during 10 epochs with the training data using optimizer Adam.

[67]: Coates et al. (2011), "An analysis of single-layer networks in unsupervised feature learning"

The results can be analyzed in Table IV.4, which shows the designed AE achieves a reduction in the mean squared error of about 89%. For comparison purposes, a basic AE has also been trained with Fig. IV.6 displays some of the test inputs together with their reconstruction by the network. The resulting reconstructions remove most of the noise and appear slightly softer than the originals.

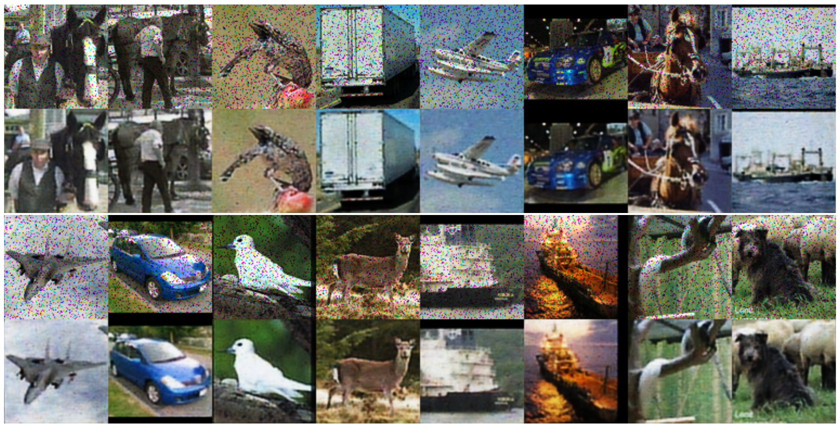


Figure IV.6: Random selection of test examples (first and third rows) and their reconstructions (second and fourth rows) via forward passes through the denoising AE.

Semantic hashing

Hashing usually refers to the process of summarizing large batches of data in smaller or simpler codes. Hashes are employed in data structures for fast search times, they can be used to find duplicates and to protect data against corruption and manipulation.

This task in particular, semantic hashing [7], involves finding binary codes which form buckets of similar data, i.e. when two data points are similar to each other, there is high probability that they will be assigned the same hash. Furthermore, if two similar data points are not hashed identically, their hashes will likely differ in only a few digits. In consequence, a way of finding instances similar to a query instance is to hash it and look for those whose hashes are the same or almost identical. This is the opposite of cryptographic hashing [80], where the likelihood of two similar entries obtaining the same hash is almost zero and there is no way of retrieving a document from its hash.

[7]: Salakhutdinov et al. (2009), "Semantic hashing"

[80]: Katz et al. (2014), *Introduction to modern cryptography*

The idea of finding semantic relations between data points is especially useful in document searches: if a query document is provided, then the search method should find those documents in the dataset which match as closely as possible. It is also of application in an image domain, where finding matching binary sequences is much more efficient than comparing two pictures [81].

[81]: Carreira-Perpinán et al. (2015), "Hashing with binary autoencoders"

The approach described in [7] uses a very simple AE architecture, with an added noise generator after the encoding which forces the encoder to polarize its outputs.

[7]: Salakhutdinov et al. (2009), "Semantic hashing"

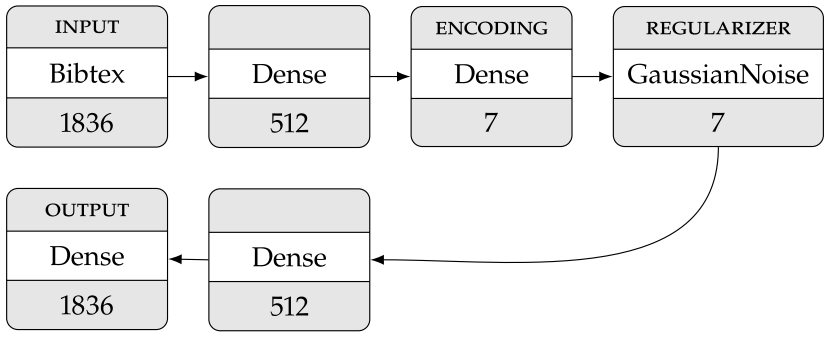


Figure IV.7: AE architecture for semantic hashing

In this case, the Bibtex dataset [68] was selected to illustrate the application. Fig. IV.7 shows the AE architecture that was defined for this purpose. The input data provides 1836 binary features which are then projected onto a smaller feature space and lastly onto a 7-dimensional encoding, which is in turn slightly corrupted before decoding. The noise introduced in the encoding during training requires it to take extreme values, for the noise not to affect the reconstruction.

[68]: Katakis et al. (2008), "Multilabel Text Classification for Automated Tag Suggestion"

In order to assess whether the trained model serves the purpose of semantic hashing, we can group all possible pairs of hashes according to their Hamming distance (e.g. 0001000 and 001001 are 1 digit away from each other, while 1010101 and 0101010 are separated by a Hamming distance of 7). Then, we measure the intercluster distance between those pairs of hashes, computed as the mean cosine distance from each instance in the first cluster to each one in the second. Assuming the clusters group similar instances, the intercluster distance should increase along with the Hamming distance. The distances for this example are illustrated in Fig. IV.8, which indeed shows simultaneous growth of both.

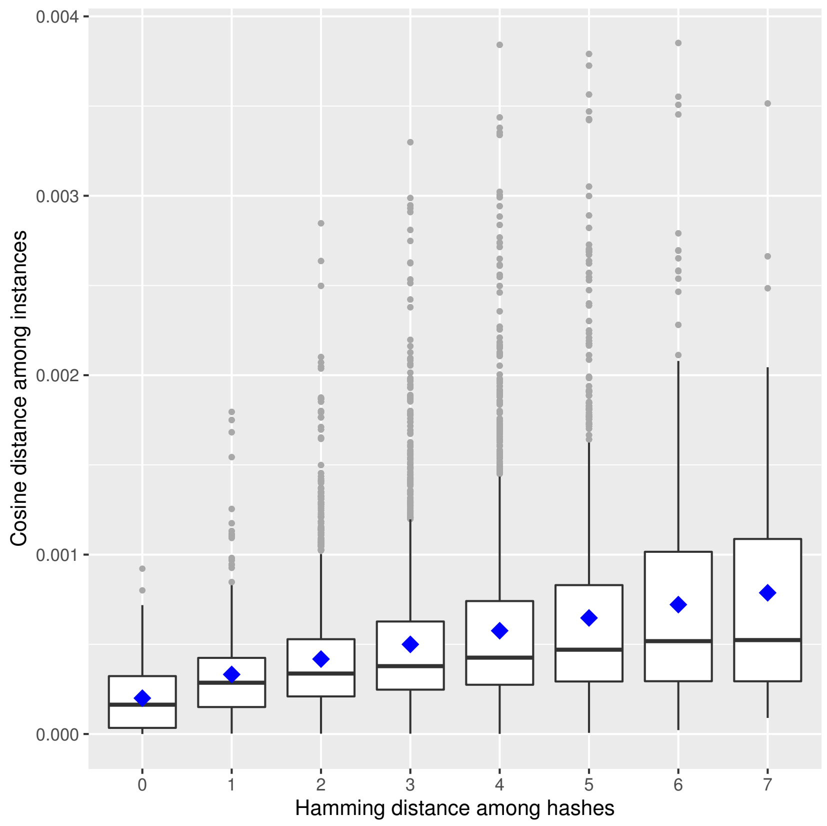


Figure IV.8: Intercluster cosine distance boxplot according to the hamming distance between hashes. Blue diamonds indicate the mean cosine distance among all pairs of clusters that differ in k digits where k is a Hamming distance. Gray dots indicate outlier cosine distances.

In addition to quantitatively evaluating the quality of the model, it can be qualitatively analyzed in order to verify whether semantic hashing indeed groups topics in similar hashes. One way of doing this is computing the term frequency-inverse document frequency index (tf-idf) [82] of the words for each cluster. This way, words that are frequent within a cluster but uncommon along the rest of the test set are considered the most relevant words. Table IV.5 shows a truncated list of hashes used by the AE to cluster documents, along with their most relevant words ranked by tf-idf.

[82]: Robertson (2004), "Understanding inverse document frequency: on theoretical arguments for IDF"

Anomaly detection

Sometimes the objective of a machine learning task is to find unusual behaviors or abnormalities in data, for example, detecting a possible security attack by analyzing server logs, or identifying rare patterns

Table IV.5: The first 15 hashes used as semantic codes for clusters found by the AE, ordered in Gray code. The most relevant words are selected according to tf-idf computed for each cluster. They show some common topics between hashes 0001110 and 0001010, and between 0000001, 0001001 and 0001000.

Hash	Relevant words
0000001	thermodynamic, transitions, induced, generalized, completely, interacting
0000011	relaxation, barrier, mainly, contribute, surfaces, rights
0000010	lipoproteins, capacity, oxidation, apo, receptor, recognized
0000110	identifying, amino, united, capable, matrix, region
0000111	carbon, storage, enzymes, assessed, notes, roles
0000101	infrastructure, configuration, challenge, location, qualitative, improvement
0000100	innovation, construction, ontologies, communities, 1999, located
0001100	mining, advances, bioinformatics, er, solved, intelligence
0001101	reuse, object, perspectives, intelligent, notes, logic
0001111	trans, reading, behavioral, cultural, 1997, gap
0001110	ss, siamese, betta, splendens, male, fighting
0001010	siamese, ss, fighting, male, display, fish
0001011	treated, barrier, combines, electrostatic, solvent, molecule
0001001	thermal, boltzmann, origin, bulk, fluctuations, disorder
0001000	numerically, temperatures, exact, magnetic, glass, zero

in medical checks. This is known as anomaly detection because the cases of interest are few in contrast to the amount of normal instances, and even in some cases there are no anomalies to train with. In this situation, a traditional classifier cannot solve the problem since it will not be able to assign a class it has not seen before.

An approach to anomaly detection without previously observed anomalous cases is to model those considered typical, and mark as anomalies those instances which do not fit the model. An AE can be used for this purpose, since it can be trained to accurately encode and reconstruct instances following a certain distribution. When the AE is feeded new instances, it is assumed that reconstruction of anomalous data will not be as accurate, since it should follow a different distribution [83–85]. More formally, the hypothesis of this methodology is that, when trained with normal data, $d(x, g(f(x)))$ will be very small when x is normal and very high when x is anomalous.

[83]: Petsche et al. (1996), "A neural network autoassociator for induction motor failure prediction"
[84]: Sakurada et al. (2014), "Anomaly detection using autoencoders with nonlinear dimensionality reduction"
[85]: Park et al. (2018), "Anomaly Detection for HTTP Using Convolutional Autoencoders"

An useful application of anomaly detection where real world data will generally lack anomalies is network intrusion [86, 87], that is, the detection of potential security attacks and malicious accesses to a server. The straightforward approach is to continuously log server accesses, and extract data from a period of time where usage has been normal. By means of these data, an AE can be trained to recognize typical usage parameters. Then, new log accesses are constantly feeded to the AE in order to predict their reconstruction

[86]: Mirsky et al. (2018), "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection"
[87]: Shone et al. (2018), "A Deep Learning Approach to Network Intrusion Detection"

error. In the case that several successive errors are much higher than the mean, an attack may be underway.

The AE used for this purpose will work as follows: the encoding layer will perform a drastic dimensionality reduction in order for it to model the most essential information from the training data, which does not include any anomaly. This should help have low error rates on normal data, similar to training instances, but very high ones on anomalous data. In general, this may not work well for uncommon, isolated anomalies, but it is useful when anomalies are several in sequence, so this strategy is especially designed for time series data.

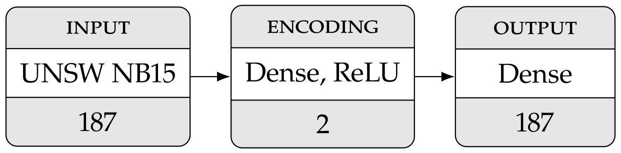


Figure IV.9: Denoising AE architecture for anomaly detection

The dataset treated in this example is UNSW-NB15 [69], which has 3 nominal variables and 42 numerical descriptors. Since AEs cannot work directly with nominal variables, these have been converted into dummy binary variables. In addition, any anomalous data from the training subset has been removed. In total, 37000 instances with 187 features are being introduced as the training input of the AE, whose architecture is shown in IV.9. The extraction of two features is sufficient to model an approximation of most of the normal data, but cannot preserve enough information for the reconstruction of most anomalies.

[69]: Moustafa et al. (2015), "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)"

The results of training this model are summarized in Fig. IV.10 and Fig. IV.11. The first is a precision-recall curve which gives details about the fraction of detections which are actually anomalies and the ratio of detected anomalies among all of them. We find that it is possible to detect more than half the anomalies without obtaining too many false alarms. Since the test dataset contains many more anomalies than normal instances and the objective is to detect abnormal sections more than to find every individual anomaly, a recall of around 50% could be enough as long as the precision is high so that few false alarms are raised.

Indeed, Fig. IV.11 graphs the reconstruction error for each test instance and shows that when an adequate threshold is chosen, anomalous sections can be easily detected with very few isolated false alarms that can be discarded. In this case, the chosen threshold is the mean reconstruction error plus 6 times its standard deviation, but it could be tuned high or low in order to adjust the sensitivity of the detection.

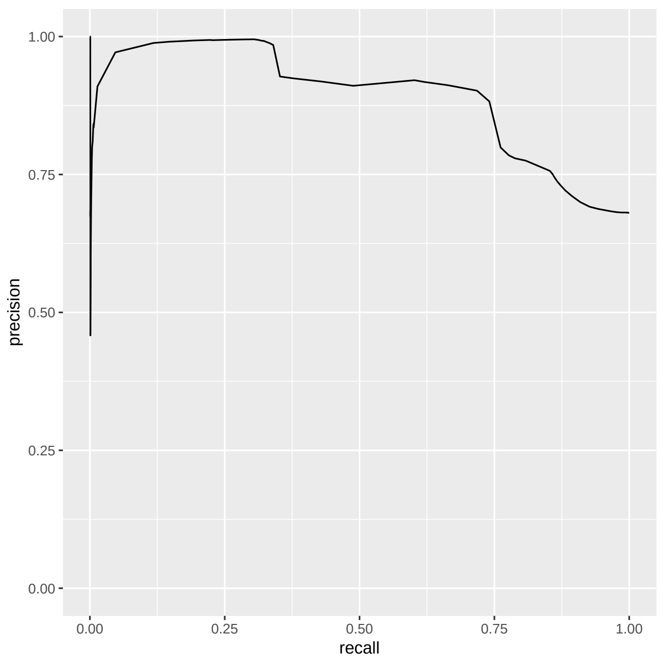


Figure IV.10: Precision-recall curve for the detection of individual anomalies in the UNSW dataset.

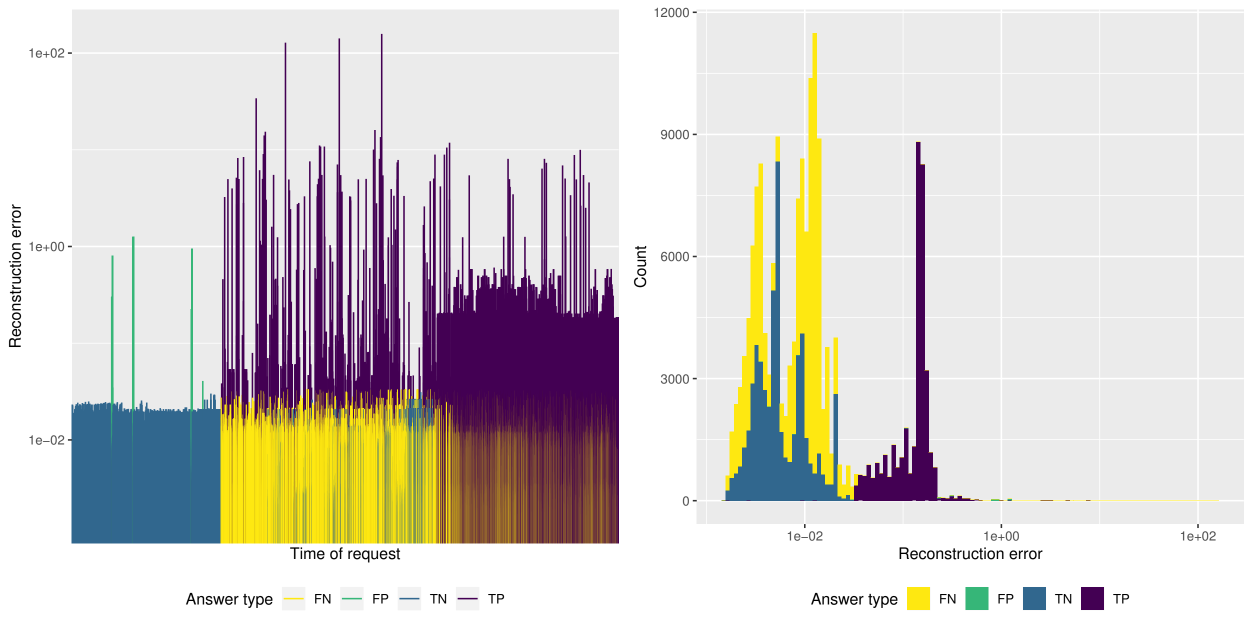


Figure IV.11: Reconstruction error of the AE during test. The graph on the left shows the reconstruction error of each request in sequence, where the detection threshold is set to the mean training error plus 6 times its standard deviation. The histogram on the right shows the amount of hits and misses according to the reconstruction error.

Instance generation

The representation learned by an AE may be useful to encode or reconstruct individual instances from a training set, but in certain cases it will be very convenient to ensure that this representation is actually attempting to perform some kind of manifold learning, mapping the feature space onto a smaller space in a way that makes sense to work with the whole encoding space. This encoding space would allow to predict a reconstruction for encodings that do not come from an instance in the original feature space, and still produce a coherent result. For instance, an useful application would be to generate new images of faces similar to those in a training set but not identical to any of them. This is usually harder to achieve with simple operations such as interpolation, because they would compute many images that do not represent faces.

There are several variants of AEs that can fulfill this purpose, namely variational [60], adversarial and contractive AEs. Variational as well as adversarial AEs force a prior distribution in the encodings in different ways, which allows to sample new instances by taking points from this space and projecting them onto the original feature space via reconstruction (g). The contractive AE, on the contrary, only imposes a regularization which promotes instances to be mapped to encodings near their neighbors. This helps the autoencoder perform transformations that find manifolds in the data, since local structure is preserved. The manifold can then be traversed in order for the decoder to generate new instances.

[60]: Kingma et al. (2013), "Auto-encoding variational bayes"

Variational AEs are stochastic in the sense that they do not map each instance to a single point in the embedding space, but a distribution instead. This is usually a normal distribution, defined by its mean and standard deviation. Then, a reconstruction is produced by sampling that distribution and propagating the results through the decoder network. The objective function in this AE combines the clustering behavior of the reconstruction loss function with a regularization loss which forces the distribution to be as similar as possible to, generally, a multivariate unit Gaussian. This helps the AE extract a very compact representation which only preserves the necessary information to provide a reconstruction of the input.

In this example, a variational AE following the structure in Fig. IV.12 is trained to generate human faces that do not belong to any person, since they will not be present in the training dataset. The input data used during training belong to the AT&T faces dataset[‡], also known as Olivetti faces dataset. The resulting model can be sampled by feeding arbitrary values to the generator component, which then outputs previously unseen images. Fig. IV.13 shows some representative examples of the generated faces using this AE.

[‡] AT&T faces dataset is available at `https://www.openml.org/d/41083`.

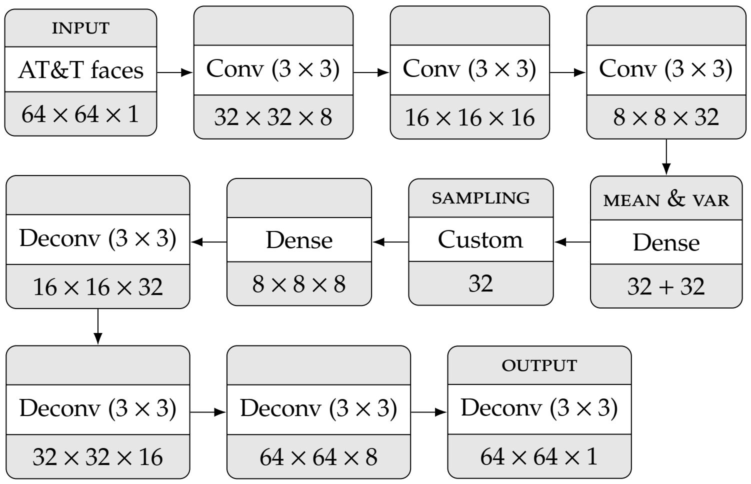


Figure IV.12: Variational AE architecture for instance generation. The sampling layer draws a sample from the vector of normal distributions with means and variances given by the previous layer.

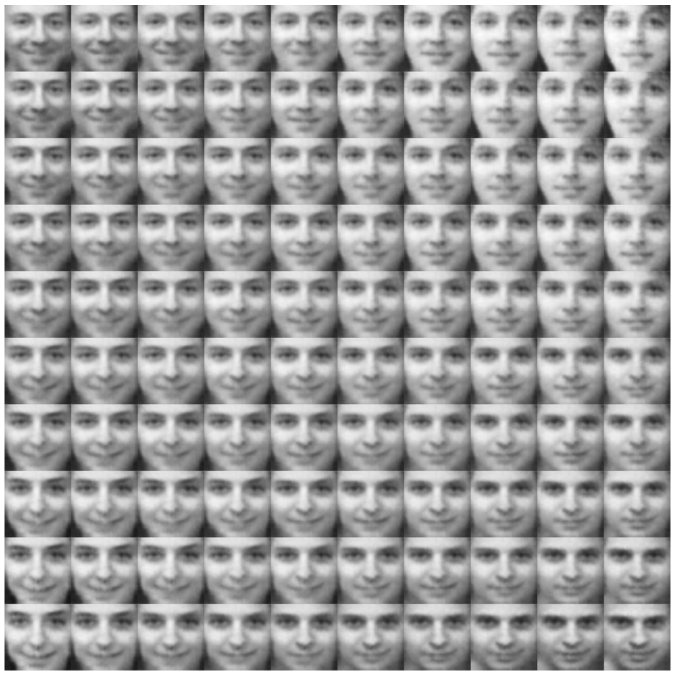


Figure IV.13: Faces sampled from the encoding space of a variational AE, using interpolations between the projections of images in the original dataset

Other applications

Apart from the previous selection of applications approached with representation learning techniques based on AEs, there are many other situations where AEs can be applied to extract features from data. The following are learning applications present in the literature that fell out of the scope of this article.

Image superresolution

This problem consists in building a high resolution image from a low resolution sample, such as a thumbnail. By using an AE trained with low resolution images and another with the high resolution ones, a map can be trained from the first encoding to the second [88]. This way, the encoder from the first AE can be connected to the decoder from the second AE and the resulting network can be fine-tuned. During prediction it suffices with feeding a low resolution image through the new network, which will encode it and decode it through the high resolution decoder, producing a higher quality image.

[88]: Zeng et al. (2015), "Coupled deep autoencoder for single image super-resolution"

Image compression

Images are usually compressed with algorithms designed for this specific purpose, e.g. the JPEG standard [89]. Since a compression mechanism must include a component which compresses the image and another which performs decompression, AEs can be trained in different ways to treat this problem as well [8, 90, 91], even surpassing the capacity of JPEG2000 especially at low bit rates.

[89]: Wallace (1992), "The JPEG still picture compression standard"

[8]: Theis et al. (2017), "Lossy image compression with compressive autoencoders"
[90]: Ballé et al. (2017), "End-to-end optimized image compression"
[91]: Cheng et al. (2018), "Deep convolutional autoencoder-based lossy image compression"

Transfer learning

In a transfer learning task, the learner must make use of the knowledge extracted from data in a given domain to apply it to a different domain. This may consist in using pre-trained networks with a large dataset to use them with a small dataset by a fine-tuning process. However, when labels for the large dataset are not available, the first stage will necessarily be unsupervised [51], in which case an AE can be trained and its extracted features can initialize a network for a supervised problem with a dataset from other domain.

[51]: Bengio (2012), "Deep learning of representations for unsupervised and transfer learning"

Human pose and facial features

Human pose recovery is an application specific to image and video data where people appear and the aim is to recognize the pose of each person from the visual information, i.e., to generate a skeleton describing the position and orientation of the legs, arms and the rest of the body. One of the challenges is to model this skeleton as a 3D object while images are only 2D. AEs have been used as the core of a human pose recovery model [92] for extracting an inner pose representation which then maps onto a representation of the 3D pose and is decoded as a 3D pose. This process is, in fact, achieved with two AEs, one for each inner representation required, which are then connected through the representation mapping. In a similar way, facial expression recognition aims to identify the

[92]: Hong et al. (2015), "Multimodal deep autoencoder for human pose recovery"

human emotional state from facial images. An approach based on deep sparse autoencoders [93], which are used to extract robust and discriminative features, has been developed to tackle this task.

[93]: Zeng et al. (2018), "Facial expression recognition via learning deep sparse autoencoders"

3D shape learning

Extracting features from three-dimensional shapes usually has a high computational cost but it is fundamental for tasks such as 3D object retrieval and matching. There are several AE-based models for automatic feature extraction that can help model this type of data [94–96]. These range from simple stacked AEs to combinations of convolutional AEs and extreme learning machines. In general, retrieving similar objects to a given input consists in encoding the input and comparing the result to the codes of known objects in order to find the nearest or most similar ones.

[94]: Zhu et al. (2016), "Deep learning representation using autoencoder for 3D shape retrieval"
[95]: Wang et al. (2016), "An efficient and effective convolutional autoencoder extreme learning machine network for 3d feature learning"
[96]: Leng et al. (2015), "3D object retrieval with stacked local convolutional autoencoder"

Recommender systems and tagging systems

Recommender systems are filters that seek to predict user preferences for products, taking into account previous choices or ratings. Collaborative filters for recommendation combine the information of different users to build predictions. In [97], a collaborative variational AE for recommendation is developed. It models the implicit relationships among items and users by making use of a shared latent representation and the variational regularization. A task similar to recommendation is tagging, since tags can be ranked for an item according to its similarity to other items. AEs have been also used as the core of tagging systems [98] using denoising AEs and relational denoising AEs.

[97]: Li et al. (2017), "Collaborative variational autoencoder for recommender systems"

[98]: Wang et al. (2015), "Relational stacked denoising autoencoder for tag recommendation"

IV.5 Challenges for autoencoder progress and prospects on explainability

Along this section, several difficulties and consequences of using AEs in machine learning are explored. Some brief comments are provided beforehand on the current state of explainability and transparency in artificial intelligence (AI), in order to understand how they could affect the way AEs are designed and used.

First, we introduce the most popular approaches to finding transparent and explainable machine learning models. Later, we develop on the ways AEs can help build interpretable solutions to different problems, by learning disentangled and fair features.

State and prospects on explainability

Explainable AI [26] encompasses many concepts around the idea that people should be able to understand how trained machine learning models work and why they make their decisions.

[26]: Arrieta et al. (2020), "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI"

The recent surge in interest in explainable models derives from the bias found in existing models as well as the search for AI safety [99]. The first issue involves models that make decisions potentially affecting human beings and those decisions can discriminate against certain population groups, e.g. people of color or women. For instance, a prediction model for criminal recidivism was found to be heavily biased against African-American people [100]. The second concept relates to the presence of relatively autonomous agents, such as robots, which execute the actions computed by a machine learning model. These models sometimes find unexpected ways to optimize their reward function (reward hacking) [101], even without completing the objective or having other potential consequences (side effects).

[99]: Amodei et al. (2016), "Concrete Problems in AI Safety"

[100]: Angwin et al. (2016), "Machine bias"

[101]: Bird et al. (2002), "The evolved radio and its implications for modelling the evolution of novel sensors"

Model transparency

The issues above reflect the fact that we should not completely trust trained models unless we can comprehend the ways they are making decisions and predictions. This has attracted the interest of researchers, domain experts and users to more explainable models and strategies to explain *black-box* models, a category which includes most deep learning techniques.

The variety of algorithms to fit machine learning models to data presents a tradeoff between performance and explainability: usually, a simpler, more explainable model is less performant than an opaque model. As a consequence most simple models, such as decision trees, rule-based learners and k-nearest neighbors, are considered transparent, since they provide an interpretable behavior out of the box.

When a model is not transparent enough, there are two main ways to approach explainability: one can use different post-hoc explainability approaches, or modify the model to facilitate our understanding of its decision process. Some new models derived from deep feature learners are designed to improve transparency and be more self-explanatory.

One way deep learning models can increase transparency is by highlighting which input features are causing their predictions. For example, attention-based models [102] have an embedded scoring technique which highlights the zones in the input that are being taken into account to make predictions. This works for image classification and object detection [103] as well as for document processing [104].

[102]: Cao et al. (2015), "Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks"
[103]: Wang et al. (2017), "Residual attention network for image classification"
[104]: Yang et al. (2016), "Hierarchical attention networks for document classification"
[105]: Chen et al. (2019), "This looks like that: deep learning for interpretable image recognition"

A different proposal for transparent image classification is a convolutional neural network-based classifier which identifies prototypes in similar images [105], that is, it provides examples on images of the same class that justify the prediction.

Explainability techniques

When an opaque model is used, there are still ways to improve our understanding of its inner workings or its predictions. In many cases, a post-hoc explainability technique may be applied. The different methods that can render a model more interpretable are usually categorized into two groups. They can be either model-agnostic, if they work independently of the model used, or model-specific, otherwise.

Some examples of model-agnostic approaches and tools are the following:

- Local approximations. LIME [106] this is a method which linearly performs a local approximation of a classifier or regressor, in a way which is interpretable. An AE-based variant of LIME has been developed to improve its stability [107].
- FairML [108]. The FairML toolbox can find strong dependencies between model outputs and the input features.
- Sensitivity analysis [109]. It is a computation based on the derivative of the conditional probability of not predicting a class given the input features. This defines a vector field where each vector indicates the direction an instance needs to be moved to, so as to be classified differently.
- Auditing. Trained models can be repeatedly tested against different inputs in order to analyze how the outputs are affected. These inputs, however, need to be provided according to some criteria. As a way to compute direct and indirect influence of each feature in the output of a model, there is a procedure which obscures the effect of a variable in the data [110]. It works

[106]: Ribeiro et al. (2016), ""Why Should I Trust You?": Explaining the Predictions of Any Classifier"

[107]: Shankaranarayana et al. (2019), "ALIME: Autoencoder Based Approach for Local Interpretability"
[108]: Adebayo et al. (2016), "FairML: ToolBox for diagnosing bias in predictive modeling"
[109]: Baehrens et al. (2010), "How to explain individual classification decisions"

[110]: Adler et al. (2018), "Auditing black-box models for indirect influence"

without retraining the model, and can assess the degree in which a feature is relevant to a classifier. There are several other approaches to analyzing direct influence of a feature in the output of a model [111, 112].

[111]: Henelius et al. (2014), "A peek into the black box: exploring classifiers by randomization"

[112]: Datta et al. (2016), "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems"

- Counterfactuals [113–115]. This is an approach with a similar objective to auditing but from a different perspective. Finding a counterfactual consists in detecting the smallest possible change in feature values that causes an alteration to the prediction of the model. These serve as an explanation for the "closest possible world" where the prediction would have been different, without providing further insight into the decision process.

[113]: Molnar (2019), "Interpretable Machine Learning"

[114]: Wachter et al. (2017), "Counterfactual explanations without opening the black box: Automated decisions and the GDPR"

[115]: Arrieta et al. (2020), "Plausible Counterfactuals: Auditing Deep Learning Classifiers with Realistic Adversarial Examples"

There are several specific techniques for explaining the outputs of deep learning models:

- Layer-wise relevance propagation [116]. This is a methodology for visualization of pixel-wise contributions to predictions, where classifiers are decomposed into several layers of computation, so the relevance of each pixel is found by propagating relevance backwards through the network.

[116]: Bach et al. (2015), "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation"

- Saliency map generation [117]. Saliency maps are heatmaps where the most relevant features from the input are highlighted. These are usually applied to convolutional neural networks in order to obtain the image regions that cause the output for each instance.

[117]: Simonyan et al. (2013), "Deep inside convolutional networks: Visualising image classification models and saliency maps"

- DeepLIFT [118]. This is a technique for computing relevance for each input feature to a neural network, by assigning contibution scores to each neuron according to its activation given a specific input.

[118]: Shrikumar et al. (2017), "Learning important features through propagating activation differences"

- SHAP [119]. This tool provides several model-specific techniques which find local explanations for different models based on Shapley values from game theory. In particular, it includes DeepExplainer and GradientExplainer, which apply to deep learning models.

[119]: Lundberg et al. (2017), "A Unified Approach to Interpreting Model Predictions"

- Traceability [120]. This is a more theoretical concept from the field of software development that could be applied to deep neural models. It seeks to describe how each component of a final inference model is related back to its training model, the dataset, hyperparameters and all the way up to some high level requirements on what task the model should carry out. Being able to trace every item in the development of a deep neural networks to a higher level cause could serve to ensure that all choices such as hyperparameters and architecture are well justified.

[120]: Aravantinos et al. (2018), "Traceability of deep neural networks"

Current challenges and influence in future work

As discussed in the previous section, most of the well-known explainability techniques involve analysis of features in one way or another. The contribution that AEs can provide in this field is, therefore, substantial. This is due to the fact that AEs can transform a set of highly dependent, correlated features in a different set of independent, interpretable ones, by using adequate regularizations. In this section, we comment on different ways to learn features that are *meaningful* and *fair*, and on recent developments for also improving the explainability of the feature extraction process itself.

Improving features: disentanglement and fairness

One way extracted features can improve their quality is by holding an understandable meaning by themselves, e.g. a model could train with face pictures and extract a feature for hair color, another one for nose size, etc. These new features would be much more useful than the original ones which represent individual pixels. This task is usually known as *feature disentanglement*.

Some recent AE models whose objective is to disentangle features are Total Correlation VAE [121], Wasserstein AE [122] and InfoGAN [123]. All of these are generative models, so, as a result, extracted features not only provide interpretable meaning to instances, but can also be sampled in order to generate unseen examples in a way that resembles the manipulation of existing instances: for example, a model could generate a realistic face similar to an existing image but changing blonde hair to black.

[121]: Chen et al. (2018), "Isolating Sources of Disentanglement in Variational Autoencoders"
[122]: Rubenstein et al. (2018), "Learning disentangled representations with wasserstein auto-encoders"
[123]: Chen et al. (2016), "Infogan: Interpretable representation learning by information maximizing generative adversarial nets"
[124]: Zemel et al. (2013), "Learning fair representations"

Another step forward in improving learned representations is forcing these to become fair [124], which means that the extracted features obfuscate information about membership to potentially discriminated groups, e.g. gender or ethnicity. Fairness usually applies only in contexts where model predictions affect human lives, e.g. job applications, legal proceedings, etc. A statistic can be defined to measure the discrimination of a classifier with respect to a binary variable. The objective is then to optimize a tradeoff between classification accuracy and discrimination [125].

[125]: Edwards et al. (2015), "Censoring representations with an adversary"
[126]: Madras et al. (2018), "Learning adversarially fair and transferable representations"

There already exist AE-based models for learning fair representations. In [126], an adversarial AE-based classifier is proposed where the adversary attempts to predict the sensitive (potentially discriminatory) attributes from the encoding, but its prediction ability is minimized by the AE and classifier. The objective function can be adjusted according to the desired type of fairness. Another model in [127] consists in a variational AE which disentangles sensitive information from the non-sensitive latent features and is flexible in the sense that potentially sensitive information can be retained or removed from the encoding during inference.

[127]: Creager et al. (2019), "Flexibly Fair Representation Learning by Disentanglement"

Explainable feature learning

The described approaches provide the possibility of explaining the end predictions of other models, as well as rendering them fairer. However, as has been extensely discussed in this work, the extracted features can be the actual core of a solution to many problems. As a consequence, it would be necessary as well to develop strategies which facilitate the explainability of the transformations an AE can perform in order to learn features. This is an area only explored very recently, but there are already some developments.

Variational AEs can be used to detect anomalies, similarly to the denoising AE explained in Section 12. In addition, they enable another, more explainable way of detecting anomalies: computing the gradients of the reconstruction error with respect to the inputs [128]. This allows to notice which input features are contributing to the error, and to cluster anomalies according to this same criterion.

[128]: Nguyen et al. (2019), "GEE: A gradient-based explainable variational autoencoder for network anomaly detection"

A different approach to improving the explainability of the embedding consists in restricting the operations each neuron performs to just logical AND/OR operators [129], which limits the origin of each extracted feature to a relatively simple logical combination of the input features, thus facilitating its interpretability.

[129]: Al-Hmouz et al. (2019), "Logic-driven autoencoders"

Influence in future works

There is currently much to be researched in the area of explainable AEs as well as AEs which help explain other models by extracting better features. The current trends focus especially on generative models such as variational AEs for these purposes, and will probably continue to do so, even if some diversification is achieved as new works appear.

The adaptability of AEs to many different problems, illustrated in previous sections, together with the possibility of producing interpretable and fair features, may lead to an increase in usage of these models throughout all kinds of machine learning applications.

In our future work, we intend to approach explainable feature learning in the context of AEs, that is, find AE-based models that extract features and at the same time provide an understandable meaning to the mapping from the original features to the encoded ones. Ideally, an explainable feature learner should not be restricted to one end application, but could be used for many purposes, as common AEs already can.

IV.6 Conclusions

Throughout this text, we have summarized the traditional alternatives for learning representations, the origins and essential characteristics of AEs, including how to introduce certain behaviors into the coding layer.

Later, we have thoroughly examined several case studies of AE applications in unstructured data as well as images and sequences: data visualization, image denoising, semantic hashing, anomaly detection and instance generation. Other applications have also been briefly discussed: image superresolution, image compression, transfer learning, human pose recovery and recommender systems.

An introduction to the state of the art in explainable AI and its application to the field of AEs has been provided as well. AEs have notoriously contributed to the areas of feature disentanglement and fair representations, and there have been some recent developments on explainable feature learning as well.

We can conclude that AEs are a versatile framework for solving a wide variety of problems where a central task is to learn representations of the data. They can adapt to a given problem in structure as well as in the objective they optimize. This way, if the solution to a problem can be modeled with a transformation of the feature space onto another space, there will be many instances where the parameters of the transformation can be learned by an AE.

Acknowledgments: D. Charte is supported by the Spanish Ministry of Science, Innovation and Universities under the FPU National Program (Ref. FPU17/04069). This paper is partially supported by the Spanish National Research Projects TIN2015-68854-R and TIN2017-89517-P and the project DeepSCOP Ayudas Fundación BBVA a Equipos de Investigación Científica en Big Data 2018.

References

[1] David Charte, Francisco Charte, Maria J del Jesus, and Francisco Herrera. “An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges”. In: *Neurocomputing* 404 (2020), pp. 93–107. doi: 10.1016/j.neucom.2020.04.057.

[2] Pedro Domingos. “A Few Useful Things to Know About Machine Learning”. In: *Communnications of the ACM* 55.10 (Oct. 2012), pp. 78–87. doi: 10.1145/2347736.2347755.

[3] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. “How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity”. In: *ACM Computing Surveys (CSUR)* 52.5 (2019), p. 107. doi: 10.1145/3347711.

[4] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Vol. 72. Springer, 2015.

[5] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. “Supervised machine learning: A review of classification techniques”. In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), pp. 3–24.

[6] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. “Data Clustering: A Review”. In: *ACM Computing Surveys* 31 (1999), pp. 264–323. doi: 10.1145/331499.331504.

[7] Ruslan Salakhutdinov and Geoffrey Hinton. “Semantic hashing”. In: *International Journal of Approximate Reasoning* 50.7 (2009), pp. 969–978. doi: 10.1016/j.ijar.2008.11.006.

[8] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. “Lossy image compression with compressive autoencoders”. In: *Fifth International Conference on Learning Representations*. 2017.

[9] Jun Deng, Zixing Zhang, Erik Marchi, and Björn Schuller. “Sparse autoencoder-based feature transfer learning for speech emotion recognition”. In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. IEEE. 2013, pp. 511–516. doi: 10.1109/ACII.2013.90.

[10] Junyuan Xie, Linli Xu, and Enhong Chen. “Image denoising and inpainting with deep neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 341–349.

[11] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*. Vol. 207. Springer, 2008.

[12] Manoranjan Dash and Huan Liu. “Feature selection for classification”. In: *Intelligent data analysis* 1.1-4 (1997), pp. 131–156. doi: 10.1016/S1088-467X(97)00008-5.

[13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828. doi: 10.1109/TPAMI.2013.50.

[14] Ian T Jolliffe. *Principal component analysis*. Springer, 1986.

[15] Joshua B Tenenbaum, Vin de Silva, and John C Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323. doi: 10.1126/science.290.5500.2319.

[16] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[17] Mikhail Belkin and Partha Niyogi. "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation". In: *Neural Computation* 15 (2003), pp. 1373–1396. DOI: 10.1162/089976603321780317.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), p. 436. DOI: 10.1038/nature14539.

[19] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1. DOI: 10.1038/323533a0.

[20] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407. DOI: 10.1007/978-1-4612-5110-1_9.

[21] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159.

[22] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[23] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *Third International Conference on Learning Representations*. 2015.

[24] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-RMSProp". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[25] David Charte, Francisco Charte, Salvador García, María J del Jesus, and Francisco Herrera. "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines". In: *Information Fusion* 44 (2018), pp. 78–96. DOI: 10.1016/j.inffus.2017.12.007.

[26] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58 (2020), pp. 82–115. DOI: 10.1016/j.inffus.2019.12.012.

[27] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. *Dimensionality reduction: a comparative review*. Tech. rep. 2009.

[28] Ronald A. Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[29] Haitao Zhao, Shaoyuan Sun, Zhongliang Jing, and Jingyu Yang. "Local structure based supervised feature extraction". In: *Pattern Recognition* 39.8 (2006), pp. 1546–1550. DOI: 10.1016/j.patcog.2006.02.023.

[30] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. "Adversarial autoencoders". In: *Fourth International Conference on Learning Representations*. 2016.

[31] Jidong Zhao, Ke Lu, and Xiaofei He. "Locality sensitive semi-supervised feature selection". In: *Neurocomputing* 71.10-12 (2008), pp. 1842–1849. DOI: 10.1016/j.neucom.2007.06.014.

[32] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 6* 2.11 (Nov. 1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[33] Harold Hotelling. “Analysis of a complex of statistical variables into principal components”. In: *Journal of educational psychology* 24.6 (1933), p. 417. DOI: 10.1037/h0071325.

[34] Ian T Jolliffe. “Principal Component Analysis and Factor Analysis”. In: *Principal component analysis*. Springer, 1986, pp. 115–128. DOI: 10.1007/978-1-4757-1904-8.

[35] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. “Nonlinear component analysis as a kernel eigenvalue problem”. In: *Neural computation* 10.5 (1998), pp. 1299–1319. DOI: 10.1162/089976698300017467.

[36] Bernhard Schölkopf. “Statistical Learning and Kernel Methods”. In: *Data Fusion and Perception*. Ed. by Giacomo Della Riccia, Hans-Joachim Lenz, and Rudolf Kruse. Vienna: Springer Vienna, 2001, pp. 3–24. DOI: 10.1007/978-3-7091-2580-9_1.

[37] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. Colorado University at Boulder, Department of Computer Science, 1986.

[38] Geoffrey E. Hinton. “Training Products of Experts by Minimizing Contrastive Divergence”. In: *Neural Computation* 14.8 (Aug. 2002), pp. 1771–1800. DOI: 10.1162/089976602760128018.

[39] John W Sammon. “A nonlinear mapping for data structure analysis”. In: *IEEE Transactions on computers* 100.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems 25*. 2012, pp. 1097–1105.

[41] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[42] Teuvo Kohonen. “The self-organizing map”. In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480. DOI: 10.1109/5.58325.

[43] Pasi Koikkalainen and Erkki Oja. “Self-organizing hierarchical feature maps”. In: *1990 IJCNN international joint conference on neural networks*. IEEE. 1990, pp. 279–284. DOI: 10.1109/IJCNN.1990.137727.

[44] Jürgen Schmidhuber, Martin Eldracher, and Bernhard Foltin. “Semilinear predictability minimization produces well-known feature detectors”. In: *Neural Computation* 8.4 (1996), pp. 773–786. DOI: 10.1162/neco.1996.8.4.773.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Deep Learning”. In: http://www.deeplearningbook.org. MIT Press, 2016. Chap. Deep generative models, pp. 651–716.

[46] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *Science* 313.5786 (2006), pp. 504–507. DOI: 10.1126/science.1127647.

[47] Nianyin Zeng, Zidong Wang, Hong Zhang, Weibo Liu, and Fuad E Alsaadi. “Deep belief networks for quantitative analysis of a gold immunochromatographic strip”. In: *Cognitive Computation* 8.4 (2016), pp. 684–692.

[48] Nianyin Zeng et al. “An improved particle filter with a novel hybrid proposal distribution for quantitative analysis of gold immunochromatographic strips”. In: *IEEE Transactions on Nanotechnology* 18 (2019), pp. 819–829.

[49] Mark A Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE journal* 37.2 (1991), pp. 233–243. DOI: 10.1002/aic.690370209.

[50] Erkki Oja. "Data compression, feature extraction, and autoassociation in feedforward neural networks". In: *Artificial neural networks* 1 (1991). Ed. by Teuvo Kohonen, K. Mákisara, O. Simula, and J. Kangas, pp. 737–745.

[51] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 17–36.

[52] Dana H Ballard. "Modular Learning in Neural Networks". In: *AAAI*. 1987, pp. 279–284.

[53] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554. DOI: 10.1162/neco.2006.18.7.1527.

[54] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[56] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. "Sparse deep belief net model for visual area V2". In: *Advances in neural information processing systems 20*. 2008, pp. 873–880.

[57] Andrew Ng et al. "Sparse autoencoder". In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[58] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. "Contractive auto-encoders: Explicit invariance during feature extraction". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 833–840. (Visited on 08/28/2017).

[59] Salah Rifai et al. "Higher order contractive auto-encoder". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, pp. 645–660. DOI: 10.1007/978-3-642-23783-6_41.

[60] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[61] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103. DOI: 10.1145/1390156.1390294.

[62] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of machine learning research* 11 (2010), pp. 3371–3408.

[63] Yu Qi, Yueming Wang, Xiaoxiang Zheng, and Zhaohui Wu. "Robust feature learning by stacked autoencoder with maximum correntropy criterion". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 6716–6720. DOI: 10.1109/ICASSP.2014.6854900.

[64] Weifeng Liu, Puskal P. Pokharel, and Jose C. Principe. "Correntropy: A localized similarity measure". In: *IEEE International Joint Conference on Neural Networks, 2006. IJCNN*. IEEE, 2006, pp. 4919–4924. DOI: 10.1109/IJCNN.2006.247192.

[65] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems 27*. 2014, pp. 2672–2680.

[66] Fuzhen Zhuang, Xiaohu Cheng, Ping Luo, Sinno Jialin Pan, and Qing He. "Supervised representation learning: Transfer learning with deep autoencoders". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015, pp. 4119–4125.

[67] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 215–223.

[68] I. Katakis, G. Tsoumakas, and I. Vlahavas. "Multilabel Text Classification for Automated Tag Suggestion". In: *Proceedings of the ECML/PKDD 2008*. 2008, pp. 75–83.

[69] N. Moustafa and J. Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: *2015 Military Communications and Information Systems Conference (MilCIS)*. Nov. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[70] Martín Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[71] François Chollet and contributors. *Keras API Documentation*. https://keras.io/api/. 2022.

[72] Ian Jolliffe. "Principal Component Analysis". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096. DOI: 10.1007/978-3-642-04898-2_455.

[73] Wenchao Yu et al. "Embedding with autoencoder regularization". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 208–223. DOI: 10.1007/978-3-642-40994-3_14.

[74] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[75] Junhua Li, Zbigniew Struzik, Liqing Zhang, and Andrzej Cichocki. "Feature learning from incomplete EEG with denoising autoencoder". In: *Neurocomputing* 165 (2015), pp. 23–31. DOI: 10.1016/j.neucom.2014.08.092.

[76] Warren S Torgerson. "Multidimensional scaling: I. Theory and method". In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916.

[77] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[78] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. "Speech enhancement based on deep denoising autoencoder". In: *Interspeech*. 2013, pp. 436–440.

[79] Peng Xiong et al. "ECG signal enhancement based on improved denoising auto-encoder". In: *Engineering Applications of Artificial Intelligence* 52 (2016), pp. 194–202. DOI: 10.1016/j.engappai.2016.02.015.

[80] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

[81] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. “Hashing with binary autoencoders”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 557–566. doi: 10.1109/CVPR.2015.7298654.

[82] Stephen Robertson. “Understanding inverse document frequency: on theoretical arguments for IDF”. In: *Journal of documentation* 60.5 (2004), pp. 503–520. doi: 10.1108/00220410410560582.

[83] Thomas Petsche et al. “A neural network autoassociator for induction motor failure prediction”. In: *Advances in neural information processing systems 9*. 1996, pp. 924–930.

[84] Mayu Sakurada and Takehisa Yairi. “Anomaly detection using autoencoders with nonlinear dimensionality reduction”. In: *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM. 2014, pp. 4–11. doi: 10.1145/2689746.2689747.

[85] Seungyoung Park, Myungjin Kim, and Seokwoo Lee. “Anomaly Detection for HTTP Using Convolutional Autoencoders”. In: *IEEE Access* 6 (2018), pp. 70884–70901. doi: 10.1109/ACCESS.2018.2881003.

[86] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection”. In: *Network and Distributed Systems Security (NDSS) Symposium 2018*. Internet Society, 2018, pp. 1–15. doi: 10.14722/ndss.2018.23204.

[87] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. “A Deep Learning Approach to Network Intrusion Detection”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (Feb. 2018), pp. 41–50. doi: 10.1109/TETCI.2017.2772792.

[88] Kun Zeng, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. “Coupled deep autoencoder for single image super-resolution”. In: *IEEE transactions on cybernetics* 47.1 (2015), pp. 27–37. doi: 10.1109/TCYB.2015.2501373.

[89] Gregory K Wallace. “The JPEG still picture compression standard”. In: *IEEE transactions on consumer electronics* 38.1 (1992), pp. xviii–xxxiv. doi: 10.1145/103085.103089.

[90] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. “End-to-end optimized image compression”. In: *Fifth International Conference on Learning Representations*. 2017.

[91] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. “Deep convolutional autoencoder-based lossy image compression”. In: *2018 Picture Coding Symposium (PCS)*. IEEE. 2018, pp. 253–257. doi: 10.1109/PCS.2018.8456308.

[92] Chaoqun Hong, Jun Yu, Jian Wan, Dacheng Tao, and Meng Wang. “Multimodal deep autoencoder for human pose recovery”. In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5659–5670. doi: 10.1109/TIP.2015.2487860.

[93] Nianyin Zeng et al. “Facial expression recognition via learning deep sparse autoencoders”. In: *Neurocomputing* 273 (2018), pp. 643–649.

[94] Zhuotun Zhu, Xinggang Wang, Song Bai, Cong Yao, and Xiang Bai. “Deep learning representation using autoencoder for 3D shape retrieval”. In: *Neurocomputing* 204 (2016), pp. 41–50. doi: 10.1016/j.neucom.2015.08.127.

[95] Yueqing Wang, Zhige Xie, Kai Xu, Yong Dou, and Yuanwu Lei. “An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning”. In: *Neurocomputing* 174 (2016), pp. 988–998. doi: 10.1016/j.neucom.2015.10.035.

[96] Biao Leng, Shuang Guo, Xiangyang Zhang, and Zhang Xiong. “3D object retrieval with stacked local convolutional autoencoder”. In: *Signal Processing* 112 (2015), pp. 119–128. DOI: 10.1016/j.sigpro.2014.09.005.

[97] Xiaopeng Li and James She. “Collaborative variational autoencoder for recommender systems”. In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2017, pp. 305–314. DOI: 10.1145/3097983.3098077.

[98] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. “Relational stacked denoising autoencoder for tag recommendation”. In: *Twenty-ninth AAAI conference on artificial intelligence*. 2015.

[99] Dario Amodei et al. “Concrete Problems in AI Safety”. In: *arXiv preprint arXiv:1606.06565* (2016).

[100] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. “Machine bias”. In: *ProPublica* May.23 (2016).

[101] Jon Bird and Paul Layzell. “The evolved radio and its implications for modelling the evolution of novel sensors”. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No. 02TH8600)*. Vol. 2. IEEE. 2002, pp. 1836–1841. DOI: 10.1109/CEC.2002.1004522.

[102] Chunshui Cao et al. “Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2956–2964. DOI: 10.1109/ICCV.2015.338.

[103] Fei Wang et al. “Residual attention network for image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3156–3164. DOI: 10.1109/CVPR.2017.683.

[104] Zichao Yang et al. “Hierarchical attention networks for document classification”. In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489.

[105] Chaofan Chen et al. “This looks like that: deep learning for interpretable image recognition”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8928–8939.

[106] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[107] Sharath M Shankaranarayana and Davor Runje. “ALIME: Autoencoder Based Approach for Local Interpretability”. In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2019, pp. 454–463. DOI: 10.1007/978-3-030-33607-3_49.

[108] Julius A Adebayo et al. “FairML: ToolBox for diagnosing bias in predictive modeling”. PhD thesis. Massachusetts Institute of Technology, 2016.

[109] David Baehrens et al. “How to explain individual classification decisions”. In: *Journal of Machine Learning Research* 11.Jun (2010), pp. 1803–1831.

[110] Philip Adler et al. “Auditing black-box models for indirect influence”. In: *Knowledge and Information Systems* 54.1 (2018), pp. 95–122. DOI: 10.1007/s10115-017-1116-3.

[111] Andreas Henelius, Kai Puolamäki, Henrik Boström, Lars Asker, and Panagiotis Papapetrou. “A peek into the black box: exploring classifiers by randomization”. In: *Data mining and knowledge discovery* 28.5-6 (2014), pp. 1503–1529. DOI: 10.1007/s10618-014-0368-8.

[112] Anupam Datta, Shayak Sen, and Yair Zick. "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 598–617. DOI: 10.1109/SP.2016.42.

[113] Christoph Molnar. "Interpretable Machine Learning. A Guide for Making Black Box Models Explainable". In: https://christophm.github.io/interpretable-ml-book/. 2019. Chap. 6.1. Counterfactual Explanations.

[114] Sandra Wachter, Brent Mittelstadt, and Chris Russell. "Counterfactual explanations without opening the black box: Automated decisions and the GDPR". In: *Harv. JL & Tech.* 31 (2017), p. 841.

[115] Alejandro Barredo Arrieta and Javier Del Ser. "Plausible Counterfactuals: Auditing Deep Learning Classifiers with Realistic Adversarial Examples". In: *CoRR* abs/2003.11323 (2020).

[116] Sebastian Bach et al. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PloS one* 10.7 (2015). DOI: 10.1371/journal.pone.0130140.

[117] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034* (2013).

[118] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. 2017, pp. 3145–3153.

[119] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774.

[120] Vincent Aravantinos and Frederik Diehl. "Traceability of deep neural networks". In: *arXiv preprint arXiv:1812.06744* (2018).

[121] Tian Qi Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. "Isolating Sources of Disentanglement in Variational Autoencoders". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 2610–2620.

[122] Paul K Rubenstein, Bernhard Schölkopf, and Ilya Tolstikhin. "Learning disentangled representations with wasserstein auto-encoders". In: *Proceedings of the Sixth International Conference on Learning Representations*. 2018.

[123] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems 29*. Curran Associates, Inc., 2016, pp. 2172–2180.

[124] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. "Learning fair representations". In: *International Conference on Machine Learning*. 2013, pp. 325–333.

[125] Harrison Edwards and Amos Storkey. "Censoring representations with an adversary". In: *arXiv preprint arXiv:1511.05897* (2015).

[126] David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. "Learning adversarially fair and transferable representations". In: *arXiv preprint arXiv:1802.06309* (2018).

[127] Elliot Creager et al. “Flexibly Fair Representation Learning by Disentanglement”. In: *International Conference on Machine Learning*. 2019, pp. 1436–1445.

[128] Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. “GEE: A gradient-based explainable variational autoencoder for network anomaly detection”. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2019, pp. 91–99. doi: 10.1109/CNS.2019.8802833.

[129] Rami Al-Hmouz, Witold Pedrycz, Abdullah Balamash, and Ali Morfeq. “Logic-driven autoencoders”. In: *Knowledge-Based Systems* 183 (2019), p. 104874. doi: 10.1016/j.knosys.2019.104874.

Reducing data complexity using autoencoders with class-informed loss functions

V

Source

David Charte, Francisco Charte, and Francisco Herrera. “Reducing Data Complexity using Autoencoders with Class-informed Loss Functions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: 10.1109/tpami.2021.3127698

Abstract

Available data in machine learning applications is becoming increasingly complex, due to higher dimensionality and difficult classes. There exists a wide variety of approaches to measuring complexity of labeled data, according to class overlap, separability or boundary shapes, as well as group morphology. Many techniques can transform the data in order to find better features, but few focus on specifically reducing data complexity. Most data transformation methods mainly treat the dimensionality aspect, leaving aside the available information within class labels which can be useful when classes are somehow complex.

This paper proposes an autoencoder-based approach to complexity reduction, using class labels in order to inform the loss function about the adequacy of the generated variables. This leads to three different new feature learners, Scorer, Skaler and Slicer. They are based on Fisher’s discriminant ratio, the Kullback-Leibler divergence and least-squares support vector machines, respectively. They can be applied as a preprocessing stage for a binary classification problem. A thorough experimentation across a collection of 27 datasets and a range of complexity and classification metrics shows that class-informed autoencoders perform better than 4 other popular unsupervised feature extraction techniques, especially when the final objective is using the data for a classification task.

V.1 Introduction

A classical obstacle in the field of data science is obtaining data of sufficient quality in order to extract the desired knowledge. The process of learning a model can be notably hindered by data presenting very common traits such as noise [2], outliers [3], high dimensionality [4] or complex class boundaries [5]. This results in long periods of time spent cleaning and preprocessing data [6] before the actual data mining step can even begin. Although data cleaning techniques can be of good use in order to identify and filter out noise, outliers and missing data, other aspects can be trickier to solve.

[2]: Xiong et al. (2006), "Enhancing data analysis with noise removal"

[3]: Aggarwal (2015), "Outlier analysis"

[4]: Ayesha et al. (2020), "Overview and comparative study of dimensionality reduction techniques for high dimensional data"

[5]: Ho et al. (2002), "Complexity measures of supervised classification problems"

[6]: García et al. (2015), *Data preprocessing in data mining*

Many real world situations can be modeled as supervised classification problems, those where each instance belongs to one of several classes, and the objective is to learn from the observed data from each class in order to automatically assign the corresponding class labels to new, unobserved instances. Some examples of classification problems are text categorization [7], spam filtering [8], object recognition in images [9] and automatic interpretation of medical data to facilitate diagnostics [10]. Many of these problems correspond to the simplest case, binary classification, where there are only two categories.

[7]: Yang et al. (1999), "A re-examination of text categorization methods"

[8]: Dada et al. (2019), "Machine learning for email spam filtering: review, approaches and open research problems"

[9]: Russakovsky et al. (2015), "ImageNet Large Scale Visual Recognition Challenge"

[10]: Deo (2015), "Machine learning in medicine"

One of the type of issues that is very commonly overlooked in classification problems is the complexity of data [5, 11]. Consider a clean dataset with no presence of errors or abnormalities. There can still be aspects related to the geometrical shapes and overlap among classes which can hinder the performance of a learning technique. For example, there could be no separability between classes, or even regions of the feature space with a mix of instances from different classes. In cases where separability is achievable, boundaries can present complex shapes that can be difficult for a parameterized model to fit. Figure V.1 illustrates some of these cases, more concretely, one where the features do not allow to separate the classes, and another where class boundaries are difficult to model due to there being several small groups from one class, sometimes known as small disjuncts [12], within a group of the other.

[5]: Ho et al. (2002), "Complexity measures of supervised classification problems"

[11]: Lorena et al. (2019), "How complex is your classification problem? a survey on measuring classification complexity"

[12]: Weiss (1995), "Learning with rare cases and small disjuncts"

An additional hindrance that frequently occurs in data mining scenarios is associated to the representation of instances, to the features themselves [13]. These can be typically seen as observed outcomes of underlying factors that cause them and, as a result, are not always the ideal representation of the data. This can depend on the objective task and the learning method to be used. For example, a pixel-based representation for images can be ideal for a convolutional

[13]: Van Der Maaten et al. (2009), *Dimensionality reduction: a comparative review*

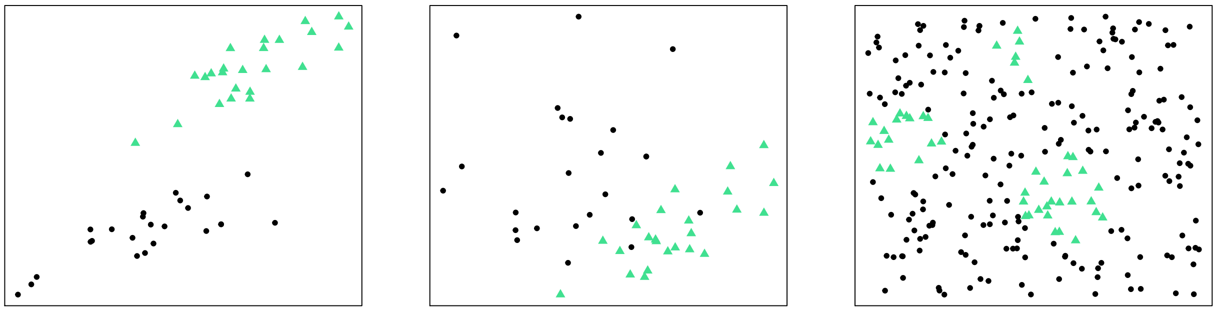


Figure V.1: Different situations relating to class complexity. The graph on the left shows separable classes, the middle one is an example where classes are not separable and the one on the right shows separable classes with complex boundaries (small disjuncts).

neural network to perform classification, but may be difficult for a lazy learning method to process.

When data have some kind of complexity, it can affect the performance of machine learning methods and these are usually not able to overcome the issue by themselves. Instead, a preprocessing step can transform data aiming to find a better representation which makes it easier to categorize points. Operating with features for this purpose is a task known as feature extraction or feature learning. There exists a wide variety of approaches to feature extraction [14], including linear transformations, manifold learning and neural network-based models. However, very few of them take class complexity into account and, as a result, extracting quality features with a specific strategy to reduce this complexity is still an important challenge.

[14]: Bengio et al. (2013), "Representation learning: A review and new perspectives"

In particular, autoencoders (AEs) [15] are neural networks specifically designed to extract features from the data. These are typically trained to reconstruct the input at its output, feeding the data through several layers which impose some kind of restriction or bottleneck in order to find more appropriate representations along the way. AEs can also be easily restricted or adapted in order to promote certain kinds of transformations and encodings, for example, finding sparse variables which only take high values for a small number of instances [16].

[15]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

[16]: Ng (2011), "Sparse autoencoder"

This work makes use of the well-known technique for regularizing the behavior of an AE, applied in this case to achieve class complexity reduction. Aiming to transform features onto a more useful space with special attention to class complexity, three concrete models that use different criteria are proposed. The bases for these are: Fisher's discriminant ratio, the Kullback-Leibler divergence (KLD) and least-squares support vector machines (LSSVMs). The new models have been tested against well-established feature extraction methods within a binary classification pipeline.

In summary, the main contributions of this paper are the following:

- New AE-based models able to learn from input features as well as binary class labels, specifically the following three variants:
  - Scorer, a model which enables separability among classes by means of the Fisher's discriminant ratio.
  - Skaler, a model that receives feedback from the KLD and can thus provide features where positive and negative instances belong to very different distributions.
  - Slicer, an extended AE using a LSSVM in order to simultaneously evaluate a simple linear classifier and assess the adequacy of the new features for classification.
- A thorough experimentation across 27 cases and 11 evaluation measures, focusing on different complexity rates and classification performance, and against 4 other well-known feature extraction methods.
- A comparison between the most interpretable complexity metrics and several evaluation metrics for classifiers, revealing which of the complexity metrics are better predictors of classification performance.

As an important conclusion after the experimental analysis of the newly proposed models, we must point out that they can be trained to generate better features for the purposes of classification than other popular feature extraction methods.

The rest of this paper is organized as follows. Section V.2 describes the current state with respect to available complexity measures and techniques to overcome complexity in data. Next, Section V.3 introduces our proposals and provides all the details about their inner workings. Section V.4 explains the details of the experimentation process, while Section V.5 discusses the results. Lastly, conclusions and final comments are provided in Section V.6.

V.2 State of the art in complexity reduction

A dataset can present many different problems that may drive it to be considered difficult or complex to classify. Initially, a possible measure of complexity could be the error rate of the classifier itself. However, the objective of this work is identifying difficult datasets to treat them before learning a classifier. For this reason, we rely on other metrics which aim to characterize the complexity of supervised problems.

Sources of difficulty

Ho and Basu [17] identify three possible sources of difficulty: (1) class ambiguity, (2) boundary complexity and (3) sample sparsity and feature space dimensionality. The first applies to the circumstances where classes cannot be distinguished using the given features, either because they provide insufficient knowledge about the problem or because the classes are not well defined. The second source refers to the situation where classes are interleaved or not easily separable. In these cases, the complexity can be measured attending to class overlap and class separability as well as geometry, topology and density of manifolds. The last category covers issues with the structure of the sample, whether it is complete enough and the amount of variables the classifier needs to work with.

[17]: Basu et al. (2006), *Data complexity in pattern recognition*

Complexity measures

In order to quantitatively assess how complex a dataset is, a wide variety of complexity metrics have been proposed over the years [5, 11]. The following sections briefly describe the most relevant approaches to measure complexity, paying special attention to the metrics that will be applied throughout the experimentation. Each metric is abbreviated according to the original nomenclature in [5] and [11].

[5]: Ho et al. (2002), "Complexity measures of supervised classification problems"

[11]: Lorena et al. (2019), "How complex is your classification problem? a survey on measuring classification complexity"

[5]: Ho et al. (2002), "Complexity measures of supervised classification problems"

[11]: Lorena et al. (2019), "How complex is your classification problem? a survey on measuring classification complexity"

Class overlap

Geometrical complexity in a dataset can be characterized in several ways. One approach is to measure the overlap in feature values among different classes. Each feature can be assessed as to how much it contributes to distinguishing the classes. In this case, measures usually focus on binary problems. The following measures follow this approach:

Maximum Fisher's discriminant ratio (F1) Fisher's discriminant ratio is a measure of class overlap, based on the simplest statistics for a distribution, mean and standard deviation. Higher values of this metric mean lower levels of overlap. The maximum over all features is taken as a measure of the class separation in a dataset.

Maximum feature efficiency (F3) Feature efficiency is calculated as the proportion of examples that can be unambiguously classified by a simple threshold, that is, they lie outside an overlapping region. This rate gives an idea of the usefulness of a given feature when attempting to classify every instance in the dataset. The maximum of this ratio across all features is known as F3.

Class separability and nonlinearity

Instead of measuring the importance of the overlapping regions in features, an alternative approach is to look for complexity of the boundary separating classes, that is, its ability to actually isolate both classes and its nonlinearity. Several measures have been developed regarding the shape and separation degree of classes.

[18]: Steinwart et al. (2008), *Support vector machines*

Linear classifier error (L2) Linear separability of classes is the core of a branch of classification methods, support vector machines (SVM) [18]. In its simplest form, a SVM is a binary classifier that attempts to find the hyperplane which best separates both classes. Its training error can be used as a metric to characterize the separability (or lack thereof) of a dataset.

Linear classifier nonlinearity (L3) Describing the shape of the regions occupied by each class can also contribute to learning about the complexity of the data. In particular, this measure tackles nonlinearity, i.e. the smoothness of the decision boundary of a classifier, which can be detected by interpolating pairs of points of the same class to extract a test set and computing the classification error for this new set.

Neighborhoods and morphology

The previous traditional measures for data complexity come from a statistical or geometrical point of view. Other metrics look at how instances are located around each other, so they study local behavior instead of global properties.

1-NN classifier error (N3) Similarly to the L1-L3 measures, which make use of a simple classifier in order to measure complexity, this metric performs a leave-one-out validation of a nearest neighbor classifier, that is, it checks the class of every instance according to the nearest one, and measures complexity as the error rate obtained.

Recently, some new metrics have been proposed that attempt to describe data complexity from the perspective of data morphology [19]. These are based on the Pure Class Cover Catch Digraph (P-CCCD) classification method [20].

[19]: Pascual-Triana et al. (2021 (forthcoming)), "Revisiting Data Complexity Metrics Based on Morphology for Overlap and Imbalance: Snapshot, New Overlap Number of Balls Metrics and Singular Problems Prospect"

[20]: Manukyan et al. (2016), "Classification of Imbalanced Data with a Geometric Digraph Family"

P-CCCD creates a collection of balls that cover the feature space so that each ball only contains points from the same class. The process consists in choosing a ball so that it is centered in a point of the target class and is the largest possible ball that does not any point of the other class. This is repeated until all points of that class are in at least one ball, producing a cover which is not necessarily optimal but is a good approximation.

Morphology-based complexity metrics are inspired by this algorithm in the sense that they look for a ball cover of all points where balls only contain points from one class, and then perform some computations according to the number of balls created. The main metrics are as follows:

Total number of balls (ONB_{tot}) This measure counts the total number of balls required to produce the cover. If b^+ balls are needed to cover all positive instances and b^- are necessary for the negative points, it is calculated as

$$ONB_{\text{tot}} = \frac{b^+ + b^-}{n} \ . \tag{V.1}$$

Average number of balls (ONB_{avg}) It averages the amount of balls used to cover the points of each class. In a binary classification environment, the definition would just be the sum of the balls-to-points ratios, divided by 2:

$$ONB_{\text{tot}} = \frac{\frac{b^+}{n^+} + \frac{b^-}{n^-}}{2} \ . \tag{V.2}$$

These metrics can turn into a very general way of describing the geometrical complexity of the classes, since the shape of the balls depends on the distance chosen (e.g. Euclidean, Manhattan or the maximum distance). The mechanism for covering the feature space attempts to use as few balls as possible to cover all points. If the dataset can be covered by a few large balls, then its complexity will be low, but if many small balls are needed, it means that many little clusters of different classes are near each other, and the complexity is thus high. Both ONB_{tot} and ONB_{avg} are, as a result, higher the more complex the data is. The difference between them is that ONB_{avg} gives the same weight to all classes, while ONB_{tot} does not distinguish classes but gives the same weight to all instances.

Feature space dimensionality

One of the main issues that occur in many datasets and has been tackled from many perspectives is dimensionality. Dimensionality refers to the number of variables where each instance takes values. High dimensionality has long been considered a problem for classification algorithms, known as curse of dimensionality [21]. It is not directly related to the way classes interact with each other, but a high number of features can hinder the performance of a classifier with a dataset that is otherwise not considered complex, due to the fact that most distance metrics lose meaning when measuring across many variables.

[21]: Aggarwal et al. (2001), "On the surprising behavior of distance metrics in high dimensional space"

Dimension can be measured in absolute terms, but the complexity that derives from it is also related to the number of instances in the dataset. Two problems with the same number of features are not equally complex if the first one has 10 times more instances than the other. As a result, an instances-to-features rate (T2) can be considered a complexity metric that can give a better account of this relation.

Other models for complexity

When trying to reduce the complexity present in a dataset, one can take complexity measures into account for evaluation purposes, and use other ways of modeling complexity when training and performing data transformations. For instance, considering that each class presents different distributions across each variable, some similarity or dissimilarity metrics for distributions could be used.

The Kullback-Leibler divergence is a well-known measure of how a distribution differs from another one, it is asymmetric as it usually compares a distribution coming from data with a distribution representing a model or theory. If these are defined on a discrete probability space $\mathcal{X}$, then the divergence is formulated as

$$D_{\mathrm{KL}}(p\|q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \, . \tag{V.3}$$

This quantity could provide an intuition on how two distributions are overlapping or separated. It is higher the more different the distributions are. One way to retrieve a symmetric value out of it is to add the Kullback-Leibler divergence of the distributions in reverse order: $D_{\mathrm{KL}}(p\|q) + D_{\mathrm{KL}}(q\|p)$. For both measures, the chosen distribution when applying them to class separability could be a Bernoulli distribution for each feature in the encoding, so that their values are considered either high or low. If we model all features at the same time, a categorical distribution could be employed. Assuming a binary classification problem, we could measure the dissimilarity of the distribution corresponding to positive instances against the distribution of negative instances, which would provide a sense on how easy it is to differentiate them.

Reducing complexity in datasets

There are several approaches to complexity reduction in datasets. This section provides a general overview of the different aspects that can be treated and techniques for doing so.

Dimensionality of data has been one of the most diversely tackled issues. A multitude of methods exist in the literature, ranging from simple feature selection to nonlinear feature learning. A thorough review of all these can be found in [6], but we enumerate and describe the main ones below.

[6]: García et al. (2015), *Data preprocessing in data mining*

However, there are other emerging methods that may be able to modify other aspects of the data and reduce complexity along the way. Some of those are distance metric learning methods.

Feature selection

Assuming that not every variable has the same relevance for the purposes of classification, an initial approach to dimensionality reduction can be to simply discard some of them, retaining only the ones that help the classifier the most. This process is known as feature selection [6]. Of course, there exist a plethora of criteria that can apply for this purpose.

[6]: García et al. (2015), *Data preprocessing in data mining*

Filters This variety of techniques is mostly founded on statistical and information theory measures, such as the joint mutual information, the conditional mutual information, the Kullback-Leibler divergence or minimum-reduncancy-maximum-relevance. The objective is to quantify the utility of each variable and keep only the most useful ones. In fact, some approaches take class separability into account as well [22, 23].

[22]: Zhang et al. (2013), "Divergence-based feature selection for separate classes"
[23]: Wang (2008), "Feature selection with kernel class separability"

Wrappers Another way of looking at feature selection is shaping it as an optimization problem, finding an adequate fitness function, typically the performance of a classifier, and making use of one of the many existing metaheuristics available, for instance, genetic algorithms, simulated annealing or particle swarm optimization, to name a few.

Embedded methods Some classifiers have built-in feature selection, so that they only look at the information provided by the most relevant variables. These are usually decision trees like C4.5 [24].

[24]: Quinlan (1993), *C4.5: Programs for Machine Learning*

Linear feature extraction

Another way of reducing the number of variables is to attempt to summarize most of the information of the original variables in a smaller set of new variables, which emerge as linear transformations of the original ones.

Principal component analysis (PCA) [25, 26] PCA is a well-studied technique that solves the problem of obtaining features which retain the maximum possible variance while being uncorrelated to each other. It also allows to recover the original data from the projected points while losing the minimum amount of information as measured by the mean squared error.

[25]: Pearson (1901), "LIII. On lines and planes of closest fit to systems of points in space"
[26]: Jolliffe (1986), *Principal component analysis*

Linear discriminant analysis (LDA) [27] This is a supervised method able to extract linear combinations of features which achieve good class separation. Under assumptions of normality, independence and homoscedascity, it can project the data onto a space consisting of new coordinates that best discriminate the classes. Its main drawback is that the number of resulting variables is completely determined by the number of classes. More specifically, for a problem with c classes, LDA will output a space of $c-1$ linear combinations

[27]: Fisher (1938), "The statistical utilization of multiple measurements"

of the original variables. There is a recent generalization of LDA which claims to solve its stability issues and achieve better class separation through a maximum margin criterion [28].

[28]: Li et al. (2006), "Efficient and robust feature extraction by maximum margin criterion"

Factor analysis [29] This technique assumes, unlike PCA, that a series of hidden factors are generating the observed data by means of linear combinations. The number of underlying factors is lower than the number of observed variables, and they are assumed to have zero mean and unit covariance (i.e. the identity matrix).

[29]: Jolliffe (1986), "Principal Component Analysis and Factor Analysis"

Nonlinear feature extraction

The most advanced methods for dimensionality reduction base their new variables on nonlinear transformations of the original ones.

A lot of these techniques can be grouped in a concept known as manifold learning, since they attempt to find structure for a manifold where most of the data lie, and thus transform each data point onto its coordinates on that manifold.

[30]: Borg et al. (2005), *Modern multidimensional scaling: Theory and applications*

Multidimensional scaling (MDS) [30] This is a classical methodology that has served as basis for several other algorithms as well. Its objective is to compute new coordinates for data points while preserving distances among them as faithfully as possible. Instead of having points as inputs, it only receives the pairwise distances themselves, and minimizes a loss function which helps the model obtain coordinates for each point, creating a space where the given distances are maintained.

[31]: Tenenbaum et al. (2000), "A global geometric framework for nonlinear dimensionality reduction"

Isomap [31] This method extends metric MDS in order to find coordinates that describe the actual degrees of freedom of the data while preserving distances among neighbors and geodesic distances between the rest of points. Isomap constructs a neighborhood graph where each edge is weighted according to the Euclidean distance among vertices, then uses this to compute geodesic distances instead of using straight lines. These new distances are potentially higher than the Euclidean but help capture more information about the manifold.

[32]: Roweis et al. (2000), "Nonlinear dimensionality reduction by locally linear embedding"

Locally linear embedding (LLE) [32] The objective of LLE is similar to that of the previous techniques, but with a different approach to preserving the local structure. It finds a linear combination which describes each point from its neighbors. Once its coefficients have been computed, LLE optimizes the coordinates for a lower-dimensional space so that they fit the same expressions.

[33]: Van der Maaten et al. (2008), "Visualizing data using t-SNE."

t-stochastic neighbor embedding (t-SNE) [33] This is a technique specially oriented for visualization, so it finds specially attractive low-dimensional projections of the data. It consists on assigning, for each pair of points, the probability that one point would choose the other as its nearest neighbor if neighbors are computed according to Gaussian distributions centered on each point. t-SNE then defines a low-dimensional mapping that tries to preserve these probability scores.

[15]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"

Autoencoder networks (AE) [15] AEs are neural network models which reconstruct the input at their output, using some kind of bottleneck in between so as to learn useful information from the data. We explain AEs in further detail in Section V.3.

Distance metric learning

[34]: Suárez et al. (2018), "A tutorial on distance metric learning: Mathematical foundations, algorithms and software"

Distance metric learning [34] is an area of machine learning dedicated to learning distances from datasets. These distances are built to better represent the similarities and differences among examples than standard distances, such as the Euclidean distance.

In a supervised learning context, the problem of learning a distance can be formulated as follows:

$$\operatorname*{argmin}_{d\in\mathcal{D}} l(d,S,D), \text{ where} \tag{V.4}$$

$$S = \{(x_i,x_j)\in\mathcal{X}\times\mathcal{X} \,:\, y_i = y_j\} \tag{V.5}$$

$$D = \{(x_i,x_j)\in\mathcal{X}\times\mathcal{X} \,:\, y_i \neq y_j\} \tag{V.6}$$

and l is a loss function that determines the fitness of a distance to describe the similarities and differences provided as sets S and D.

Some of the dimensionality reduction methods mentioned above can be also seen as distance metric learning techniques, but there are more algorithms which can learn distances. Some of them are addressed at improving the performance of k-nearest neighbors, and others are based on information theory. Among the most relevant are: NCA [35], LMNN [36], NCMML [37] and NCMC [37].

[35]: Goldberger et al. (2004), "Neighbourhood components analysis"

[36]: Weinberger et al. (2009), "Distance metric learning for large margin nearest neighbor classification."

[37]: Mensink et al. (2013), "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost"

[37]: Mensink et al. (2013), "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost"

Current limitations

Many of the complexity reduction methods explained above tackle complexity only partially or from a limited perspective. In most of the cases, tha main focus is reducing dimensionality. This can help model data when good quality coordinates are found, but may discard useful information present in the class labels.

Some of the available methods consider class separability when selecting features [22, 23] but they do not generate new features, and can capture only a partial view of the whole feature space as a result.

[22]: Zhang et al. (2013), "Divergence-based feature selection for separate classes"

[23]: Wang (2008), "Feature selection with kernel class separability"

In summary, there is an unexplored possibility of advanced feature extraction techniques which perform nonlinear transformations of variables in order to find spaces where classes are further away and easier to identify.

V.3 Autoencoders for complexity reduction

The objective of this work is to develop strategies that address data complexity in a more complete way, that is, working with transformations of all available features and incorporating class complexity measures to acquire information from the class labels. The result is a collection of models that are based on AEs because they are very versatile deep learning architectures, able to transform the data in diverse ways according to their loss function. Our hypothesis is that when the loss function takes data complexity into account, then the AE will have more information to work with and will generate better features than other feature extraction methods.

In order to provide an indication on data complexity to a loss function, it is necessary to look for computationally simple ways of calculating a penalty that points the training method in the right direction. This problem can be approached from several possible perspectives, including the integration of a complexity metric, a measure of distribution dissimilarity or a linear separation method.

This section details the theoretical underpinings of our proposals: Scorer, Skaler and Slicer. First, some basic notions of AEs help establish a starting point for these new models. Next, the added penalties for Scorer and Skaler are explained. Lastly, the necessary modifications and computations needed for Slicer are shown as well.

Autoencoder fundamentals

A neural AE [15, 38] is generally a symmetrical neural network trained to reconstruct the inputs at its output. The composition of layers up to the middle one computes a new representation of the input data where some traits may be induced: lower dimension, sparsity, or robustness against noise, for example. The feature transformation is learned by means of a training process that optimizes the reconstruction error as well as, potentially, other penalties allowing the introduction of those specific aspects.

[15]: Charte et al. (2018), "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines"
[38]: Bengio (2012), "Deep learning of representations for unsupervised and transfer learning"

A simple AE models the reconstruction problem as a deterministic function given by the composition of an encoder f and a decoder g. When an instance is feeded to the model, the encoder transforms it to a vector located within the encoding space, and the decoder maps this vector to the original feature space.

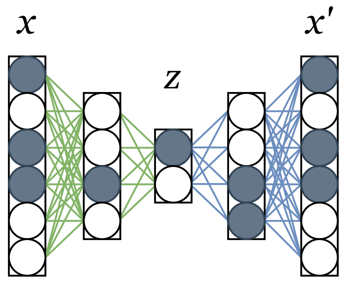


Figure V.2: The essential structure of an AE implemented as a fully connected feed-forward neural network, composed of an encoder f and a decoder g. The training loss of this model is measured as the distance d between the input x and its reconstruction $x' = (g \circ f)(x)$.

Consider the diagram in Fig. V.2. During training, mini-batches of samples are propagated through the network. The AE is evaluated according to the average distance between original and reconstructed samples. Its weights are iteratively modified in order to minimize this distance. There are two typical dissimilarity metrics for this purpose:

- Mean squared error: defined as the average of squared errors. If x and x' are a training sample and its reconstruction, it is expressed as:

$$\mathcal{L}(x, x') = \frac{1}{n}\sum_{i=1}^{n}(x_i - x'_i)^2 \tag{V.7}$$

- Cross entropy: this measure is effective when modeling data where values lie in the $[0, 1]$ interval, since it is usually implemented as the cross entropy of two Bernoulli distributions. The formulation is as follows:

$$\mathcal{L}(x, x') = -\sum_{i=1}^{n} x_i \log x'_i + (1 - x_i)\log(1 - x'_i) \tag{V.8}$$

In general, any kind of measure that indicates the difference among two data points of the same type can be used. For certain types of structured data, such as images or sequences, specific reconstruction errors may also apply. For instance, a perceptual loss [39] can be very fitting for image reconstruction, since it focuses more in the appearance of the image instead of trying to accurately recover each individual pixel, which can lead to softer and blurrier images.

[39]: Johnson et al. (2016), "Perceptual losses for real-time style transfer and super-resolution"

Once one of these dissimilarity metrics is chosen, the loss function of the AE can be defined:

$$J(\theta; S) = \sum_{(x,y)\in S} \mathscr{L}(x, (g \circ f)(x)) \,, \tag{V.9}$$

where θ holds the parameters of the network, and thus determines f and g, and S is a set of training instances.

Diverse kinds of regularizations can be applied to the loss function with the objective of adjusting the behavior of the AE, such as sparsity, contraction or variational inference. Each of these result in a slightly different AE variant with its own applications. Although these and several other regularizations help build better feature spaces, to the best of our knowledge there is no AE variant focusing on enabling class separability or reducing data complexity yet.

AEs are generally trained with common neural network optimizers, such as stochastic gradient descent [40] or Adam [41]. They decide how to update the parameters in an iterative process which computes the gradient of the loss function via backpropagation [42].

[40]: Robbins et al. (1951), "A stochastic approximation method"

[41]: Kingma et al. (2015), "Adam: A method for stochastic optimization"

[42]: Rumelhart et al. (1986), "Learning representations by back-propagating errors"

Regularizing autoencoders with label information

As described above, a basic autoencoder is trained using a loss function which evaluates the distance between the input feature vector and its reconstruction through the network. It is clear, by its definition, that instance labels are not used at all to compute the loss function, nor does the AE receive this information as input. This has its advantages and shortcomings. A benefit is that one may train AEs using unlabeled data and obtain valuable knowledge as a result. This allows for their use in several widespread applications [43], such as anomaly detection, data denoising, synthetic instance generation and semantic hashing.

[43]: Charte et al. (2020), "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges"

One possible drawback, when applying AEs to classification problems, is that they will extract features that may or may not help distinguish the classes, since they are not provided with the labels. However, this is not applicable to all AE-based models, since some of them can take the class label into account when computing the encoded representation, either directly as an input layer to the network, or indirectly, by informing the loss function.

A common way to modify the behavior of the training process and improve the solutions is to add a penalty term Ω to the loss function promoting certain aspects of the encoding or reconstruction mappings. This penalty may be dependent on the weights of the network or the resulting codes. It is added with a weight coefficient λ in order to adjust its importance with respect to the standard reconstruction error:

$$J(\theta; S) = \sum_{(x,y)\in S} \mathcal{L}(x, (g \circ f)(x)) + \lambda\Omega(\theta; S) \tag{V.10}$$

For instance, one well-known regularization consists in penalizing high levels of simultaneous activations within the codes. This, usually called a sparsity regularization [16], helps maintain a low number of active neurons in the encoding for each sample.

[16]: Ng (2011), "Sparse autoencoder"

In our case, the objective is that the resulting feature transformation helps better separate different classes, so the loss function should receive some kind of label information in order to be able to learn from it. The procedure can thus be similar to a penalty modification, but using the class label within the penalty term Ω.

There could be numerous ways of analyzing the relation of codes and classes. For example, trying to optimize a complexity measure or maximizing the difference among class distributions, as well as wrapping a simple classifier so as to assess the quality of the features. These are the main ideas behind our three proposals:

- Scorer, an AE model with a Fisher's discriminant ratio-based penalty. Its objectives may be collaborating or in opposition, but it needs to find a balance between good instance reconstructions and low class overlap.
- Skaler, an AE using the KLD to separate class distributions. The encodings are modeled as a categorical distribution and the model attempts to maximize the divergence among the distribution of positive instances and that of negative instances. This should draw them apart from each other.
- Slicer, an AE which internally trains a linear least-squares support vector machine. The internal classifier need not be perfect, but it helps the model analyze how easy it is to classify the instances using the generated features. The objective, in this case, is to maximize the linear separation of both classes.

Along the rest of this section, each one of these AE-based models is thoroughly described.

Scorer

The first of our approaches to complexity reduction is to directly employ one of the complexity metrics as penalty, assuming that, if an AE is able to optimize this metric for a given dataset, the resulting representation will be less complex than the original. For this purpose, Fisher's discriminant ratio has been selected, as it is simple enough to be computed on the fly during training. The result is an AE which performs supervised class overlap reduction, or Scorer for short.

In order to introduce a complexity penalty based on Fisher's discriminant ratio, we consider the average of the discriminant ratios of each feature. This is different to the complexity measure commonly known as maximum Fisher's discriminant ratio or F1, which instead calculates the maximum of those ratios. In this case, we chose the average because it should provide better gradients in order to optimize the objective. This was corroborated by a preliminary experimentation.

The following equations formally define the complexity penalty computed within Scorer, N^+ denoting the amount of positive examples and N^- the number of negative ones. First, we define the necessary terms for the mean of each variable for positive instances and the same for negative instances.

$$\mu_j^+ = \frac{1}{N^+} \sum_{(x,+1)\in S} f(x)_j, \quad \mu_j^- = \frac{1}{N^-} \sum_{(x,-1)\in S} f(x)_j, \tag{V.11}$$

Next comes the standard deviation for each variable and for each class, calculated as the mean squared value minus the square of the mean:

$$\sigma_j^+ = \left[\frac{1}{N^+} \sum_{(x,+1)\in S} f(x)_j^2\right] - (\mu_j^+)^2 \tag{V.12}$$

$$\sigma_j^+ = \left[\frac{1}{N^-} \sum_{(x,-1)\in S} f(x)_j^2\right] - (\mu_j^-)^2 \tag{V.13}$$

This allows to put together an expression for the average Fisher's discriminant ratio, which is introduced in the loss function in a way that ensures its value to be between 0 and 1:

$$F = \frac{1}{n_f} \sum_{j=1}^{n_f} \frac{(\mu_j^+ - \mu_j^-)^2}{\sigma_j^+ + \sigma_j^-}, \quad \Omega(\theta; S) = \frac{1}{1+F} \tag{V.14}$$

The penalty term, drawing inputs from the encoding layer and the class label, is simply computed at the end of each training and added to the loss function, multiplied by a weight hyperparameter λ so as to balance it with the reconstruction objective. Figure V.3 extends the basic AE diagram with these new components in order to illustrate how Scorer differs from the basic model.

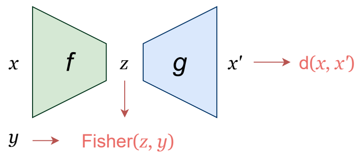


Figure V.3: Schematic illustration of the Scorer model. The average Fisher discriminant ratio of the encoded class distributions contributes to the training loss.

Skaler

The next step in inducing a class-separating behavior in an AE is to use information theory-based measures. In this case, the AE is not forced to directly optimize a complexity metric. Instead, it receives information about the current relation among class distributions, and is assessed according to the similarity of those.

Although cross entropy is the conventional measure for classification loss, we refrain from using it as a penalty because the objective is not to directly classify, thus concentrating all instances on one of two points, but to provide a representation that better clusters examples.

Skaler is a supervised feature extraction model with a KLD-based penalty for class separation. As explained above, the KLD gives an asymmetric view on how two distributions are different. In this case, the objective is to maximize the difference among the distribution of encodings belonging to the positive class and those belonging to the negative class. A schematic view is provided in Figure V.4.

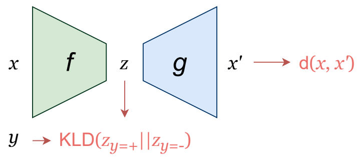


Figure V.4: Schematic illustration of the Skaler model. The KLD between the positive and the negative-class encodings contributes negatively to the training loss.

Both positive and negative encodings can be modeled as following categorical distributions, if we assume that each feature in the encoding can have a high (1) and a low (0) state and that the highest feature is the one that matters. This is a simplification but it helps build a KLD-based formulation that is easy to implement and able to train successfully. Then, the sample space of each distribution consists of events associated to each feature, indicating whether that feature is the highest. If $P_+(j)$ denotes the probability that the j-th feature is highest for positive instances and $P_-(j)$ does the same for the negative class, the KLD-based penalty function would be as follows:

$$\Omega(\theta; S) = -\sum_{j=1}^{d} P_+(j) \log \frac{P_+(j)}{P_-(j)} \tag{V.15}$$

Now, in order to compute valid probabilities for each case of the categorical distribution out of the encoding generated by the AE, we take the mean of each variable in a vector and perform the softmax activation function, obtaining a vector of values summing 1, thus representing a distribution. The j-th component of that vector corresponds to the probability that the j-th feature is high for any given data sample:

$$P_+(j) = \text{softmax}\left(\frac{1}{N^+} \sum_{(x,+1)\in S} f(x)\right)_j \tag{V.16}$$

$$P_-(j) = \text{softmax}\left(\frac{1}{N^-} \sum_{(x,-1)\in S} f(x)\right)_j \tag{V.17}$$

Some preliminary tests revealed that it is easy for this penalty to force encodings onto a single class-dependent value for any inputs. It was observed, however, that maximizing the entropy of the encoding variables helped prevent this issue, so it is added as a negative term to the penalty in the implementation.

Slicer

The third proposal of this work goes a bit further than the two previous ones, since it incorporates not only a different penalty function, but also additional learnable parameters.

This alternative regularization is inspired on least-squares support vector machines (LSSVM) [44]. These models attempt to learn the hyperplane which best separates both classes, but the difference between them and traditional SVMs lies on the objective function. For both models, the linear (non-kernelized) version of the classifier optimizes parameters w and b of the hyperplane $w^T x + b$. However, the functions that both models minimize are different. In the case of the LSSVM, the parameters are fitted to optimize the following expression:

[44]: Suykens et al. (1999), "Least squares support vector machine classifiers"

$$\frac{1}{2}\|w\|^2 + \frac{\beta}{2} \sum_{(x,y)\in S} \left(1 - y\left(w^T x + b\right)\right)^2 \tag{V.18}$$

The idea behind our model is to find a representation which facilitates the task of fitting a linear classifier. The resulting model is an AE for supervised linear classifier error reduction, hereinafter called Slicer.

In order for the model to compute the linear classifier objective function, we add trainable weights w and b to the computation graph of the neural network. These are used to get the output of a linear SVM, allowing thus to train the SVM and use it as a penalty to modify the behavior of the encoder at the same time:

$$\Omega(\theta; S) = \frac{1}{2}\|w\|^2 + \frac{\beta}{2}\sum_{(x,y)\in S}\left(1 - y\left(w^T f(x) + b\right)\right)^2, \qquad \text{(V.19)}$$

where β is just a hyperparameter weighting the importance of the LSSVM loss, and w and b are updated by the model after each epoch, just like the rest of neural network weights. In this case, the value of y is 1 for the positive class and -1 for the negative one.

Figure V.5 illustrates how the AE is modified using the LSSVM objective function, taking the encoding $z = f(x)$ as input for the LSSVM and using a prediction $p = w^T z + b$ to calculate its loss.

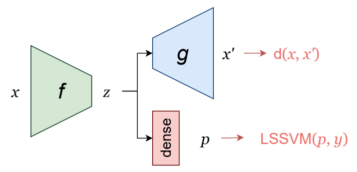


Figure V.5: Schematic illustration of the Slicer model.

The result is a model that simultaneously trains a very simple classifier on the encoded data and uses its objective in order to find better representations. Our assumption is that there will exist some level of collaboration between both models and this will help the new features become more practical for classifiers to use.

V.4 Experimental setup

Our proposals have been tested to verify their performance in reducing the complexity of data according to some measures, as well as in generating feature spaces where binary classification is easier. This section first goes through the materials for the experiments: data, methods and metrics, and then provides details on the implementations of the newly proposed models.

The experiments that were performed in order to analyze whether using class information in an AE provides an advantage include a broad range of datasets as well as several well-established methods for comparison purposes. The following sections explain the data, compared methods and evaluation metrics used along the experimentation.

Data

The methods have worked with a collection of 27 datasets from several sources with varying dimensionalities. Thirteen of them originally have binary classes, six derive from the individual labels in a multilabel dataset, five are “grouped” binarizations where several classes are taken as the positive class and the rest as the negative one, and two originate from one-vs-all scenarios where only one arbitrary class is chosen as the positive one.

The grouped binarizations have been chosen so as to present a binary scenario that would make sense with each of the problems posed by the datasets: distinguishing vowels from consonants (when the original label was the letter), odd from even handwritten digits, walking from staying movement signals and two high-level categories of soil from images. One-vs-all schemes have not been performed in these cases due to the high amount of classes and resulting experiments.

The datasets are briefly described and referenced in the supplementary material.

Complexity reduction methods

In addition to our proposals Scorer, Skaler and Slicer, we have selected four dimension reduction methods which can contribute to reducing the complexity of these datasets: PCA, LLE, Isomap and basic AE. PCA is used as the baseline for dimension reduction, LLE and Isomap are selected due to their manifold learning purpose, and the basic AE serves to analyze whether our proposals improve its behavior. None of these has the capability of learning from the classes, but they do address the dimensionality problem. The objective of our experiments is, thus, to test whether class information can be useful for an automatic feature learner to retrieve better quality attributes. Please refer to Table V.1 and Section V.2 for a brief description of the idea behind each technique and a longer explanation, respectively.

Table V.1: Brief description of each method available in the experiments

Method	Description
PCA [26]	Linear variance maximization
LLE [32]	Neighborhood-based manifold learning
Isomap [31]	MDS-based manifold learning
Basic AE [45]	Neural network for data reconstruction
Skaler	Proposed AE with Kullback-Leibler-based penalty
Scorer	Proposed AE with discriminant ratio-based penalty
Slicer	Proposed AE with LSSVM-based penalty

Evaluation metrics

In order to provide different perspectives on the performance of all methods, a diverse set of evaluation metrics has been selected. The objective is to be able to analyze the possible advantages and shortcomings of each available method.

We have trained our proposals and the compared methods to reduce the dimension of the datasets to the square root of the original dimension. The feature transformation learned by each one has been used to compute a list of complexity metrics for the resulting projections. Afterwards, we have trained several simple classifiers in order to assess the ease of classification with the generated features.

In summary, the metrics used for evaluation of each complexity reduction method can be categorized into classifier-agnostic and classifier-dependent.

Classifier-agnostic metrics are some of the complexity measures discussed in Section V.2. They have been chosen essentially for their popularity and easy interpretation. Morphology-based ONB metrics have also been computed so as to verify their affinity with the actual classification performance as well.

On the other hand, a partial objective of this experimentation is to check whether the generation of new features can actually ease classification tasks if aided by class information. The logical step is thus to analyze the performance of several datasets with the resulting variables.

- F-score. Derived from precision (the ratio of instances correctly predicted as positive) and recall (the ratio of positive instances correctly detected), it is essentially the harmonic mean of both:

$$\text{F-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{V.20}$$

- Area under the ROC curve (AUC). This metric is computed as the area, out of 1, that lays under the receiver operating characteristic curve, which denotes, as the prediction threshold goes from 0 to 1, the ratio of true positives related to the ratio of false positives.
- Cohen's Kappa. It measures the level of agreement between the predictor and the ground truth, that is, the extent to which the coincidences differ from random chance (p_c). The

mathematical definition is:

$$\kappa = 1 - \frac{1 - \mathrm{Acc}}{1 - p_c}\,. \tag{V.21}$$

These evaluation metrics have been chosen over other popular ones such as accuracy or precision since they attempt to provide a better overall picture of the performance without being affected by imbalance. Some other metrics that are also considered common complexity measures are not actually classifier-independent: linear classifier error (L2), nonlinearity of linear classifier (L3) and 1NN classifier error (N3). These were previously defined in Section V.2.

Table V.2 gathers all of these selected metrics with a short interpretation of each one. In order to obtain robust values, they have been computed over a 5-fold cross validation scheme.

	Metric	Meaning
Agnostic	F1 (Fisher)	Class overlap according to mean and variance
	F3 (efficiency)	Feature ability to separate classes
	ONB (total)	Total number of balls in cover
	ONB (average)	Average number of balls in cover
Dependent	L2	Linear classifier error
	L3	Linear classifier nonlinearity
	N3	1NN classifier error
	F-score	Tradeoff between precision and recall
	AUC	Area under the ROC curve
	Kappa	Agreement between prediction and truth

Table V.2: Brief description of each evaluation metric used for the experiments, classified according to their dependency on a classifier

Parameters

The last details about the execution of the experiments are provided in Table V.3, which shows all the values for the parameters involved in the each of the different methods.

Parameter	Value
Encoding dimension	$\max\left\{\min\left\{\sqrt{d}, \frac{n}{10}\right\}, 2\right\}$
Epochs	200
Number of hidden layers	3 (1 for < 100 variables)
Activation function (AEs except Skaler)	ReLU
Activation function (Skaler)	Sigmoid
Penalty weight - Scorer	0.01
Penalty weight - Skaler	0.1
Penalty weight - Slicer	1
Reconstruction error	Cross entropy

Table V.3: Enumeration of parameters used throughout the experiments.

V.5 Experimental results

This section presents the outcomes of the experiments performed, focusing on comparing the different methods, as well as drawing conclusions from the obtained metrics and graphics.

Results

Experiments for the 27 datasets have been conducted in a 5-fold cross validation scheme. A total of 16 metrics were computed for each case, and the full results are available at the associated website*. Next, we show and analyze aggregated results and the corresponding statistical tests.

Table V.4 holds the average ranking that each dimensionality reduction method achieved for each metric throughout the dataset collection. The winning method for each row is marked in underlined bold text. The number of overall first positions in rankings is summed up and shown in the last row of the table. The first observation that can be extracted is that model Slicer turns out to be consistently superior in most metrics, resulting in a vastly higher amount of won cases than the rest.

Table V.4: Average ranking for each method in each evaluated metric. A horizontal line separates complexity metrics from classifier evaluation metrics. The best method is marked with a star ★. Those which are worse with $p < 0.05$ are marked with ×, and those which are worse with $p < 0.01$ are marked with ⊗. The total number of first positions achieved by each method is shown in the last row.

		PCA	LLE	Isomap	AE	Skaler	Scorer	Slicer
	F1	⊗ 5.885	⊗ 6.115	⊗ 5.731	⊗ 4.038	★ **1.577**	2.308	2.346
	F3	⊗ 4.250	⊗ 5.231	⊗ 4.846	⊗ 5.654	2.788	3.000	★ **2.231**
	N3	⊗ 4.692	⊗ 5.173	⊗ 5.308	× 4.269	3.654	2.577	★ **2.327**
	L2	★ **2.712**	× 4.519	3.981	⊗ 5.654	× 4.442	3.846	2.846
	L3	2.769	× 4.500	4.115	⊗ 5.500	⊗ 4.923	3.654	★ **2.538**
	ONB_{tot}	⊗ 4.481	⊗ 6.115	⊗ 4.519	⊗ 4.481	3.442	3.019	★ **1.942**
	ONB_{avg}	⊗ 4.442	⊗ 6.077	⊗ 4.519	⊗ 4.577	× 3.538	2.904	★ **1.942**
kNN	F-score	3.096	⊗ 6.923	⊗ 4.904	4.115	3.519	3.231	★ **2.212**
	AUC	3.288	⊗ 7.000	⊗ 4.750	× 4.038	3.500	3.250	★ **2.173**
	Kappa	3.096	⊗ 7.000	⊗ 4.827	× 4.038	3.558	3.346	★ **2.135**
SVM	F-score	3.788	⊗ 6.846	⊗ 4.673	3.538	3.538	2.923	★ **2.692**
	AUC	× 4.000	⊗ 6.846	⊗ 4.865	3.462	3.346	2.962	★ **2.519**
	Kappa	3.904	⊗ 6.846	⊗ 4.827	3.346	3.577	2.962	★ **2.538**
MLP	F-score	⊗ 4.077	⊗ 6.769	⊗ 3.962	⊗ 4.731	2.596	⊗ 3.865	★ **2.000**
	AUC	⊗ 4.000	⊗ 7.000	⊗ 4.058	⊗ 4.635	2.404	⊗ 3.942	★ **1.962**
	Kappa	⊗ 4.000	⊗ 7.000	⊗ 4.077	⊗ 4.596	2.558	⊗ 3.827	★ **1.942**
	wins	62	10	17	19	78	64	**188**

Looking at the table into more detail, we can observe that the three supervised AE-based methods overall reach better metrics than the traditional feature extraction techniques, especially when analyzing the resulting predictive performance of the different classifiers.

* https://ari-dasci.github.io/S-reducing-complexity/

In order to calculate which differences are significant, the Friedman's Aligned Ranks test was performed with its corresponding post-hoc test where the winning algorithm was chosen as the control for each metric. Table V.4 organizes the results of these tests, indicating which methods were significantly worse than the winner with two levels of confidence ($p < 0.05$ and $p < 0.01$). The tests allow to assess whether the values found by the rankings can be considered enough to state that two methods are performing differently. It is important to notice, however, that each statistical test had information only about a specific metric, and not the whole picture. As a result, the fact that a test does not find significant differences among two methods does not mean that they perform the same in general.

As a last summary, we have gathered all results and performed the Friedman's Aligned Ranks test in order to obtain potential global differences among methods. Although values from different metrics are mixed in these data, the test only considers rankings so it allows to extract some intuitions about the overall performance and whether different methods can be discerned from their evaluation metrics. This test is visualized in Figure V.6, where critical distances are annotated with a confidence level of 99% ($p < 0.01$).

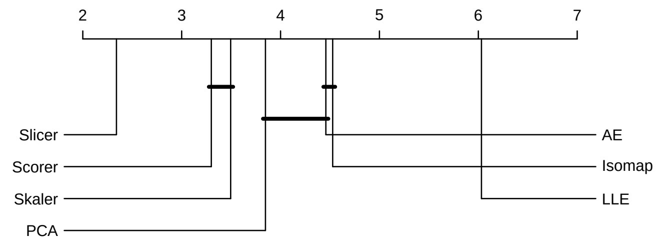


Figure V.6: Critical distance plot for all results. Horizontal lines join methods where significant differences were not detected.

The supplementary material for this paper also includes density estimation plots which provide a visual account of the overall results of each method for each metric.

Discussion

The previous results allow to notice some interesting details. One of them is the fact that the model that performed best according to the F1 metric, Skaler, is not the one that attempts to optimize that metric. This suggests that, within a neural network model, the KLD-based loss penalty is more useful for this purpose than the F1 metric itself. It is also noteworthy that Skaler, having reached the best separability metrics according to the F1 metric, ended up losing performance to Slicer when evaluated by means of actual classifiers. This could mean that the F1 metric, despite being widely known and used, is not the best estimator of classifier performance.

Looking further at the relation between the complexity metrics and the classification metrics for each classifier, it is straightforward to identify the complexity metrics that align best with classifier performance. Those would be F3, N3, L3 and the morphology-based metrics. However, the complexity metrics where rankings are most similar to the classification rankings seem to be ONB_{tot} and ONB_{avg}.

As to the comparison among complexity reduction methods, the computed rankings, critical distance plots and joyplots reveal that Slicer presents a clear advantage over the rest of methods in the majority of metrics. The individual statistical tests show many significant differences between Slicer and other methods, specifically LLE and Isomap. Some differences are also found from Slicer to PCA and AE, although not for every case. However, it is important to notice that Slicer is consistently superior to every other method across almost all metrics, something that these tests do not take into account. A more global perspective is given by the critical distance plot in Figure V.6, which does find significant superiority of Slicer over the rest of methods.

In addition to Slicer taking the top place in classification tasks, we can also see that the other supervised AEs tend to be superior than the unsupervised methods in many cases, but the differences are smaller. In fact, PCA is also competing with them when the kNN classifier is employed for classification, which is interesting considering the simplicity of the method. The standard AE, however, does not achieve very good results in comparison to the improved versions with class-informed penalties. This leads to deduce that these types of regularizations are helping differentiate our proposals from what is otherwise the exact same neural network architecture. This idea is corroborated by the overall number of wins in Table V.4, where both Skaler and Scorer achieve a higher number of wins than the rest except Slicer. Furthermore, the critical distance plot in Figure V.6 shows a significant difference from Scorer and Skaler to the classical feature extraction techniques, without being able to discard whether they perform equally.

In summary, the experimentation has shown that defining a class-based penalty in an otherwise unsupervised learning method such as an AE does help generate more useful features when the objective is training classifiers. Although the level of benefit will depend on the classifier used, any one of them will have some improvement in its performance. Of all the proposed approaches, the one that seems to retain the most information about classes within the encoded features is Slicer, which simultaneously learns the encoding and a linear classifier for the encoded variables.

V.6 Final comments

This work has introduced the concept of class complexity onto AEs that learn from class labels. If dimension reduction helps classifiers by providing more compact versions of the data, complexity reduction goes further by also improving the shape and distribution of the different classes. This concept is realized in 3 specific models with distinct behavior, Scorer, Skaler and Slicer.

Similarly to other preprocessing methods, these intend to ease a later classification task. In this case, only binary targets have been supported, since most of the complexity metrics were initially defined for binary problems. It would be possible to extend the proposed framework to multiclass or even multi-label datasets, although the penalty functions as well as the implementations would need notable modifications. A similar approach could also be followed for regression problems, where the target variable could help model the embedding space.

An additional option that may be explored is the application of a combination of penalties, since they are not necessarily exclusive, including improvements on basic AEs such as variational AEs. Some preliminary tests suggested that combining Slicer and Scorer may provide a slight improvement in classification performance, but it would need meticulous optimization of the selected penalties and their weights in order to provide promising results. Further steps would include introducing other complexity measures into the penalty function, so as to tackle other aspects of data complexity, as well as learning from just a small number of labels in a semi-supervised scenario [46].

[46]: Kingma et al. (2014), "Semi-supervised learning with deep generative models"

The experimentation developed to support the proposals has served not only to highlight the potential of Slicer as a better alternative for feature extraction, but also to notice which complexity measures are better predictors of classifier performance. In particular, morphology-based metrics like ONB_{tot} and ONB_{avg} seemed to be more aligned with classifiers.

Acknowledgments

We thank José A. Fernández for suggesting the KLD as a possible separation measure. D. Charte is supported by the Spanish Ministry of Science under the FPU program (Ref. FPU17/04069). F. Charte is supported by the Spanish Ministry of Science project PID2019-107793GB-I00 / AEI / 10.13039/501100011033. F. Herrera is supported by the Spanish Ministry of Science project PID2020-119478GB-I00 and the Andalusian Excellence project P18-FR-4961. This work is supported by the project DeepSCOP Ayudas Fundación BBVA a Equipos de Investigación Científica en Big Data 2018.

References

[1] David Charte, Francisco Charte, and Francisco Herrera. “Reducing Data Complexity using Autoencoders with Class-informed Loss Functions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: `10.1109/tpami.2021.3127698`.

[2] Hui Xiong, Gaurav Pandey, Michael Steinbach, and Vipin Kumar. “Enhancing data analysis with noise removal”. In: *IEEE Transactions on Knowledge and Data Engineering* 18.3 (2006), pp. 304–319.

[3] Charu C Aggarwal. “Outlier analysis”. In: *Data mining*. Springer. 2015, pp. 237–263.

[4] Shaeela Ayesha, Muhammad Kashif Hanif, and Ramzan Talib. “Overview and comparative study of dimensionality reduction techniques for high dimensional data”. In: *Information Fusion* 59 (2020), pp. 44–58. DOI: `https://doi.org/10.1016/j.inffus.2020.01.005`.

[5] Tin Kam Ho and Mitra Basu. “Complexity measures of supervised classification problems”. In: *IEEE transactions on pattern analysis and machine intelligence* 24.3 (2002), pp. 289–300.

[6] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Vol. 72. Springer, 2015.

[7] Yiming Yang and Xin Liu. “A re-examination of text categorization methods”. In: *SIGIR '99*. 1999.

[8] E. Dada et al. “Machine learning for email spam filtering: review, approaches and open research problems”. In: *Heliyon* 5 (2019).

[9] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115 (2015), pp. 211–252.

[10] Rahul C Deo. “Machine learning in medicine”. In: *Circulation* 132.20 (2015), pp. 1920–1930.

[11] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. “How complex is your classification problem? a survey on measuring classification complexity”. In: *ACM Computing Surveys (CSUR)* 52.5 (2019), pp. 1–34.

[12] Gary M Weiss. “Learning with rare cases and small disjuncts”. In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 558–565.

[13] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. *Dimensionality reduction: a comparative review*. Tech. rep. 2009.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828. DOI: `10.1109/TPAMI.2013.50`.

[15] David Charte, Francisco Charte, Salvador García, María J del Jesus, and Francisco Herrera. “A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines”. In: *Information Fusion* 44 (2018), pp. 78–96. DOI: `10.1016/j.inffus.2017.12.007`.

[16] Andrew Ng. “Sparse autoencoder”. In: *CS294A Lecture notes* 72.2011 (2011), pp. 1–19.

[17] Mitra Basu and Tin Kam Ho. *Data complexity in pattern recognition*. Springer Science & Business Media, 2006.

[18] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[19] José Daniel Pascual-Triana, David Charte, Marta Andrés Arroyo, Alberto Fernández, and Francisco Herrera. "Revisiting Data Complexity Metrics Based on Morphology for Overlap and Imbalance: Snapshot, New Overlap Number of Balls Metrics and Singular Problems Prospect". In: *Knowledge and Information Systems* (2021 (forthcoming)).

[20] Artür Manukyan and Elvan Ceyhan. "Classification of Imbalanced Data with a Geometric Digraph Family". In: *Journal of Machine Learning Research* (2016).

[21] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. "On the surprising behavior of distance metrics in high dimensional space". In: *International conference on database theory*. Springer. 2001, pp. 420–434.

[22] Yishi Zhang, Shujuan Li, Teng Wang, and Zigang Zhang. "Divergence-based feature selection for separate classes". In: *Neurocomputing* 101 (2013), pp. 32–42. DOI: https://doi.org/10.1016/j.neucom.2012.06.036.

[23] Lei Wang. "Feature selection with kernel class separability". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.9 (2008), pp. 1534–1546.

[24] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.

[25] Karl Pearson. "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series 6* 2.11 (Nov. 1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[26] Ian T Jolliffe. *Principal component analysis*. Springer, 1986.

[27] Ronald A Fisher. "The statistical utilization of multiple measurements". In: *Annals of Human Genetics* 8.4 (1938), pp. 376–386. DOI: 10.1111/j.1469-1809.1938.tb02189.x.

[28] Haifeng Li, Tao Jiang, and Keshu Zhang. "Efficient and robust feature extraction by maximum margin criterion". In: *IEEE transactions on neural networks* 17.1 (2006), pp. 157–165.

[29] Ian T Jolliffe. "Principal Component Analysis and Factor Analysis". In: *Principal component analysis*. Springer, 1986, pp. 115–128. DOI: 10.1007/978-1-4757-1904-8.

[30] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[31] Joshua B Tenenbaum, Vin de Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.

[32] Sam T Roweis and Lawrence K Saul. "Nonlinear dimensionality reduction by locally linear embedding". In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[33] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[34] Juan Luis Suárez, Salvador García, and Francisco Herrera. "A tutorial on distance metric learning: Mathematical foundations, algorithms and software". In: *arXiv preprint arXiv:1812.05944* (2018).

[35] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. "Neighbourhood components analysis". In: *Advances in neural information processing systems* 17 (2004), pp. 513–520.

[36] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of machine learning research* 10.2 (2009).

[37] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (2013), pp. 2624–2637. DOI: `10.1109/TPAMI.2013.83`.

[38] Yoshua Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. 2012, pp. 17–36.

[39] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European conference on computer vision*. Springer. 2016, pp. 694–711.

[40] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407. DOI: `10.1007/978-1-4612-5110-1\_9`.

[41] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations*. 2015.

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–538. DOI: `10.1038/323533a0`.

[43] David Charte, Francisco Charte, María J. del Jesus, and Francisco Herrera. "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges". In: *Neurocomputing* 404 (2020), pp. 93–107. DOI: `https://doi.org/10.1016/j.neucom.2020.04.057`.

[44] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[45] Dana H. Ballard. "Modular Learning in Neural Networks". In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*. AAAI'87. Seattle, Washington: AAAI Press, 1987, pp. 279–284.

[46] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. "Semi-supervised learning with deep generative models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 3581–3589.

Final remarks

Conclusions 5

This chapter aims to summarize the outcomes of this thesis, highlight the most relevant achievements and outline some lines of work that will be pursued next.

5.1 Summary of work

The candidate has been working under a competitive government contract for university professor training with the objective of completing this thesis. Additionally, the doctoral program at University of Granada includes a comprehensive range of lectures and courses which provide formative resources for doctoral candidates. In consequence, the work throughout this period can be categorized into three parts: research, training and teaching.

Research activity

The research developed by the candidate has been already mostly discussed in Chapter 4. The outputs have been multiple and have covered all necessary areas from the study of the state of the art, to developing a new solution to an existing problem and applying it to a specific case.

In summary, during the course of the pre-doctoral stage, the candidate has authored 11 peer-reviewed journal articles, 10 of them being in JCR journals and 8 of those in Q1 within the corresponding knowledge areas. Additionally, the candidate has attended three national AI-focused conferences and two international ones, and has presented works on all of them.

Most of the developed software to perform experimentations has been made available in the popular code repository GitHub for its use to replicate and extend them. Additionally, some of these packages conveniently allow users to apply the models to other datasets, more specifically, Ruta and the autoencoders for complexity reduction including the convolutional version of Slicer. The most relevant source code pieces are described below:

- Ruta, software for unsupervised deep architectures (associated to Article II). Homepage: ruta.software. Source code: github.com/fdavidcl/ruta.
- Autoencoder case studies (associated to Article IV). Homepage/source code: github.com/fdavidcl/ae-case-studies.
- Reducing complexity (associated to Article V). Homepage: ari-dasci.github.io/S-reducing-complexity. Source code: github.com/ari-dasci/S-reducing-complexity.
- Convolutional Slicer (associated to [1]). Homepage/source code: github.com/fdavidcl/slicer-conv.

[1]: Charte et al. (2021), "Slicer: Feature Learning for Class Separability with Least-Squares Support Vector Machine Loss and COVID-19 Chest X-Ray Case Study"

Formative activity

The different organisms of the University of Granada offer a wide variety of courses, workshops and seminars which help doctoral students complete their training and augment their skills. The candidate has leveraged this as much as time has allowed, having attended to the events detailed in Table 5.1.

Table 5.1: Relation of courses and seminars the candidate has attended.

Year	Title	Speaker(s)	Organization
2018	Iniciación a la docencia universitaria para contratados/as predoctorales FPU y FPI	multiple	UGR
2019	Open Access, Data Management y Data Protection	Pilar Rico, María José Ariza	OPI-UGR, FECYT, Biblioteca UGR
2019	Workshop de Inteligencia Artificial (Microsoft Azure)	Eduardo Matallanas	Microsoft/Plain Concepts
2020	Inverse Problems in Image Processing	Mehran Ebrahimi	CITIC-UGR
2020	¿Cómo afrontar con éxito una estancia de investigación?	Nicolás Robinson	Vicerrectorado de investigación
2021	Taller virtual de iniciación a la divulgación	Carlos Centeno, Susana Escudero	DaSCI
2022	Taller de formación Google Cloud (NLP)	Javier Martínez	Google
2020-22	DaSCI Online Seminars	multiple	DaSCI

Teaching activity

The candidate has fulfilled the teaching objectives marked by the FPU regulations: between 3 and 6 ECTS credits a year, totaling at most 18 ECTS credits. More specifically, the amount of credits has been maxed out, achieving the total of 18 credits taught across 4 years.

Taught subjects have belonged to the Computer Science department and to two different disciplines: data structures and models of computation. Table 5.2 details the subjects, groups and credits taught each year.

Year	Degree	Subject	Groups	ECTS
2019	GII	Estructuras de datos	1	1.5
2019	GIM	Estructuras de datos	1	1.5
2019	GII	Modelos de computación	1	3.0
2020	GII	Estructuras de datos	2	3.0
2020	GII	Modelos de computación	1	3.0
2021	GII	Modelos de computación	2	6.0

Table 5.2: Breakdown of the subjects where the candidate taught practical lectures during the doctoral period, detailing the number of groups and the total of ECTS credits. Abbreviations: Grado en Ingeniería Informática (GII), Doble Grado en Ingeniería Informática y Matemáticas (GIM).

In addition to the mandatory classes, during the course of the thesis, a textbook on machine learning and data science has been published for use as training material [2], as well as a 5-video course on linear algebra and dimensionality reduction, including autoencoder networks. The latter, part of a larger machine learning course titled Math-ML, has been created in collaboration with the Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI). The video playlist can be accessed at youtube.com/playlist?list=PL88MWrW4s4nf-Bc3hccxt3Att8TSS-LBn.

[2]: Charte et al. (), *Machine Learning y Ciencia de Datos con Python y R*

Collaboration with public and private entities

The doctoral period also opened opportunities to participate in collaborations established between the research group and other institutions and companies. The following is a list of these:

- Our research group has obtained several state-funded projects which involve deep learning as one of their main lines of work. Their funding allowed the group to build the necessary infrastructure to quickly develop and test models with large amounts of data.
- The candidate has collaborated with the Repsol statistics department in optimization of refinery processes. No research outputs are available due to industrial secrecy requirements.
- The candidate has participated in a two-year collaboration with the metallurgic company ArcelorMittal, on the topic of

semantic segmentation of metallographic iamges with encoder-decoder deep neural networks. A result of this project was the article [3] co-written by the UGR and ArcelorMittal teams.

[3]: Luengo et al. (2022), "A tutorial on the segmentation of metallographic images: Taxonomy, new MetalDAM dataset, deep learning-based ensemble model, experimental analysis and challenges"

- A collaboration was also established with the Hospital Universitario San Cecilio in Granada, during which the candidate looked into capsule networks and convolutional networks attempting to solve the problem of detecting the presence of COVID-19-induced lung affection. The results of this study were published in [4].

[4]: Tabik et al. (2020), "COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images"

5.2 Achieved objectives

This section explains how the different objectives posed in Section 1.4 have been tackled and completed.

Didactic resources about autoencoders

Our first objective was tied to the inexistence of a modern, exhaustive and accessible survey on the main tool, the autoencoder. The result was an extensive article (reproduced as Article I) that explained every fundamental aspect about these models, as well as provided enough detail about the main variants to be able to select an appropriate one for any given purpose.

One key contribution of this work was Section section I.6, which attempts to provide advice on which options to choose depending on the problem at hand. This article has achieved more than 100 citations according to Web of Science and more than 200 in Google Scholar, confirming that there was a need for this kind of resource on autoencoders in the research community.

Easy-to-use autoencoder implementations

The main software-related outcome of this thesis has been the R package Ruta (see Article II), a library which includes the most relevant autoencoder variants and provides easy-to-use functionalities for beginners, as well as more flexible and detailed options for more experienced users. This software is published at CRAN, a software repository for the R language with strict quality controls.

The Ruta package has totaled more than 15000 downloads just on the official RStudio CRAN mirror, averaging more than 10 downloads per day since its launch. Figure 5.1 shows daily downloads of the library according to the RStudio CRAN logs.

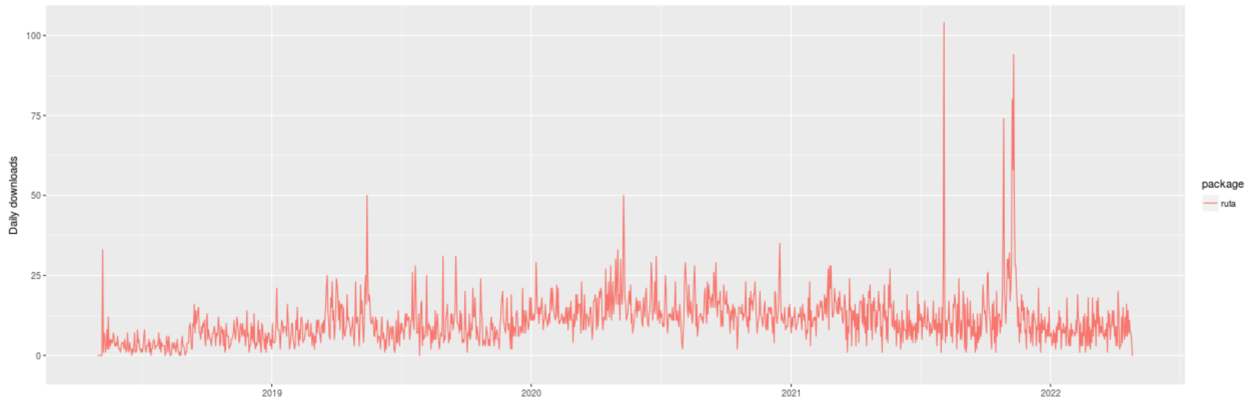


Figure 5.1: Per-day downloads of the Ruta package from the RStudio CRAN mirror.

Exploration of supervised problems

The importance of this work, materialized in Article III, resides in the fact that it helps connect all the different problem structures that can be encountered in supervised learning, from the simplest to the more general ones. More concretely, Figure III.4 visually explains how each problem relates to another by means of generalizations. This can help scientists and practitioners tackle new problem structures by means of known techniques from other problems.

Development and application of new autoencoder models

The novel contribution developed to satify this objective comprised three new penalty functions that proved to alter the behavior of an autoencoder in a way that was beneficial to undertake a posterior classification task.

The extensive experimentation showed that, out of the three methods considered, the LSSVM-inspired loss worked best and provided significant improvement in classification performance with respect to other feature learners.

When this model in particular was applied to the image classification problem given by the COVIDGR dataset, results revealed a promising line of work, as several of the selected classifiers improved their performance when learning from the features extracted by the proposed model with respect to the original features.

5.3 Future lines of work

The developed work has shown to be very promising and opens several possible routes to continue researching.

Promoting other behavior in learned representations

As was learned out of the study of autoencoder variants in Article I and the new autoencoder models from Article V, these models are flexible enough to be able to accept very diverse penalty functions which in turn influence the learned transformations in some way. Relevant uses of this property have been to learn noise-resilient representations, create probability distributions from which to sample new data, and separate points from different classes.

Leveraging this advantage of autoencoders and taking into account the diversity of supervised problems analyzed in Article III, an interesting path of research would be to conceptualize and implement new penalties which incorporate knowledge about the structure of those problems and allow this way to perform feature preprocessing on those tasks, even potentially transformations from one task type to another. For example, one specifically designed autoencoder could merge features from multi-view input samples in order to output just one feature vector, representative of the whole instance. This would effectively reduce multi-view problems to single-view ones.

Label separability in multilabel data

The early work of the candidate, although not fundamental for this thesis, revolved around learning from multilabel data. The previous experience in this specific area would help translate the newly developed solutions in Article V from the binary classification scheme to multilabel clasification, as this can be seen as a generalization of the more traditional binary case.

Approaching label separability, however, is not as straightforward as class separability in binary or multiclass problems, since several labels can co-occur in the same instance. This means that separability is no longer easy to measure: one could compute per-label data complexity measures but that would not give a complete account of how close instances with similar labelsets are. Similarly, for a model to be able to project instances to a feature space better suited for classification, it has to take all labels into account at the same time and produce variables that separate not only individual labels but relevant labelsets.

Multilabel learning being typically a harder problem than multiclass classification, there is a need for better descriptive and preprocessing tools, so a set of complexity measures and new methods for reducing complexity in datasets of this kind would be a notable advancement. Some preliminary work is currently being done by the candidate with this purpose in mind.

Synthetic instance generation for label resampling

Multilabel classification problems pose an interesting obstacle when resampling instances in order to address imbalanced labels: replicating instances with minority labels or generating similar ones usually also implies increasing the amount of instances with majority labels, since they can co-occur [5].

[5]: Charte et al. (2019), "REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization"

A novel solution to this issue would be to train generative autoencoders so as to learn a probability distribution characterized by the dataset labels. This would constitute a mechanism for creating new synthetic instances with only minority labels. Furthermore, instances generated by an autoencoder would fit better within the distribution of the original dataset than those built by pure interpolation of pairs of instances [6], like in SMOTE-based approaches.

[6]: Berthelot et al. (2018), *Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer*

References

[1] David Charte et al. “Slicer: Feature Learning for Class Separability with Least-Squares Support Vector Machine Loss and COVID-19 Chest X-Ray Case Study”. In: *International Conference on Hybrid Artificial Intelligence Systems*. Springer. 2021, pp. 305–315. doi: 10.1007/978-3-030-86271-8_26.

[2] Francisco Charte and David Charte. *Machine Learning y Ciencia de Datos con Python y R*. Krasis Press.

[3] Julian Luengo et al. “A tutorial on the segmentation of metallographic images: Taxonomy, new MetalDAM dataset, deep learning-based ensemble model, experimental analysis and challenges”. In: *Information Fusion* 78 (Feb. 2022), pp. 232–253. doi: 10.1016/j.inffus.2021.09.018.

[4] S. Tabik et al. “COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images”. In: *IEEE Journal of Biomedical and Health Informatics* 24.12 (Dec. 2020), pp. 3595–3605. doi: 10.1109/jbhi.2020.3037127.

[5] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. “REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization”. In: *Neurocomputing* 326 (2019), pp. 110–122. doi: 10.1016/j.neucom.2017.01.118.

[6] David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. *Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer*. 2018. doi: 10.48550/arxiv.1807.07543. url: https://arxiv.org/abs/1807.07543.

Alphabetical index